



```
DDDDDDDD  BBBB BBBB  GGGGGGGG  NN  NN  P P P P P P P P  NN  NN  P P P P P P P P
DDDDDDDD  BBBB BBBB  GGGGGGGG  NN  NN  P P P P P P P P  NN  NN  P P P P P P P P
DD  DD  BB  BB  GG  NN  NN  PP  PP  NN  NN  PP  PP
DD  DD  BB  BB  GG  NN  NN  PP  PP  NN  NN  PP  PP
DD  DD  BB  BB  GG  NNNN  NN  PP  PP  NNNN  NN  PP  PP
DD  DD  BB  BB  GG  NNNN  NN  PP  PP  NNNN  NN  PP  PP
DD  DD  BBBB BBBB  GG  NN  NN  P P P P P P P P  NN  NN  P P P P P P P P
DD  DD  BBBB BBBB  GG  NN  NN  P P P P P P P P  NN  NN  P P P P P P P P
DD  DD  BB  BB  GG  GG  GG  NN  NN  NN  NN  PP  NN  NN  PP
DD  DD  BB  BB  GG  GG  GG  NN  NN  NN  NN  PP  NN  NN  PP
DD  DD  BB  BB  GG  GG  GG  NN  NN  NN  NN  PP  NN  NN  PP
DD  DD  BB  BB  GG  GG  GG  NN  NN  NN  NN  PP  NN  NN  PP
DDDDDDDD  BBBB BBBB  GGGGGG  NN  NN  PP  NN  NN  PP
DDDDDDDD  BBBB BBBB  GGGGGG  NN  NN  PP  NN  NN  PP
```

```
....
....
....
....
```

```
LL  IIIIII  SSSSSSSS
LL  IIIIII  SSSSSSSS
LL  II  SS
LL  II  SS
LL  II  SS
LL  II  SS
LL  II  SSSSSS
LL  II  SSSSSS
LL  II  SS
LL  II  SS
LL  II  SS
LL  II  SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```



```
1 0001 0 MODULE DBGNPNP (IDENT = 'V04-000') =
2 0002 0
3 0003 1 BEGIN
4 0004 1
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1
30 0030 1 ++
31 0031 1 FACILITY:
32 0032 1
33 0033 1     DEBUG
34 0034 1
35 0035 1 ABSTRACT:
36 0036 1
37 0037 1     This module contains routines which collectively permform pathname
38 0038 1     parsing according to the DEBUG syntax for pathnames. The lexical
39 0039 1     scanner used by the parser is language dependent and is provided by
40 0040 1     the caller of dbg$pathname_parser.
41 0041 1
42 0042 1     The method of parsing is that of ATNs.
43 0043 1
44 0044 1     This module also contains a routine which parses the objects of a
45 0045 1     SET SCOPE command. This routine invokes the pathname parser, supplying
46 0046 1     the address of a kernel lexical scanner routine.
47 0047 1
48 0048 1 ENVIRONMENT:
49 0049 1
50 0050 1     VAX/VMS
51 0051 1
52 0052 1 AUTHOR:
53 0053 1
54 0054 1     David Plummer
55 0055 1
56 0056 1 CREATION DATE:
57 0057 1
```



DBGNPNP  
V04-000

C 1  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 2  
(1)

:	58	0058	1	:	9-SEPT-80			
:	59	0059	1	:				
:	60	0060	1	:	VERSION:			
:	61	0061	1	:				
:	62	0062	1	:	V03-004			
:	63	0063	1	:				
:	64	0064	1	:	MODIFIED BY:			
:	65	0065	1	:				
:	66	0066	1	:	John Francis	3-Jun-81		
:	67	0067	1	:				
:	68	0068	1	:	EDIT HISTORY			
:	69	0069	1	:				
:	70	0070	1	:	002	13-Mar-81	JF	Change max length to 255 (not 511)
:	71	0071	1	:	003	30-Apr-81	JF	Add support for %NAME construct
:	72	0072	1	:	004	3-Jun-81	JF	Print A\B.C rather than A\B\C!
:	73	0073	1	:	--			



```

75      0074 1  |
76      0075 1  | TABLE OF CONTENTS:
77      0076 1  |
78      0077 1  |
79      0078 1  | FORWARD ROUTINE
80      0079 1  |     DBG$NPATHNAME_PARSER,      | Entry point to parse network
81      0080 1  |     PARSE_PATHNAME,           | PN parse network
82      0081 1  |     FIRST_LINE,               | Parse first line reference
83      0082 1  |     LINE_LOOKAHEAD,           | Resolves line number -
84      0083 1  |                                | numeric scope conflict
85      0084 1  |     FIRST_LABEL,              | Parses first label reference
86      0085 1  |     LABEL_LOOKAHEAD,          | Resolves label num - n.s.
87      0086 1  |     GLOBAL_ITEM,              | Parses global id
88      0087 1  |     NUMERIC_PATHNAME,         | Parses numeric pathnames
89      0088 1  |     LINE_ITEM,                | Parses line references
90      0089 1  |     LABEL_ITEM,               | Parses label references
91      0090 1  |     QNAME_ITEM,               | Parses %NAME construct
92      0091 1  |     ID_ITEM,                  | Parses ID references
93      0092 1  |     INTEGER_ITEM,             | Parses dangling line or
94      0093 1  |                                | label numbers
95      0094 1  |     SHORT_SCOPE,              | Parses global and numeric scopes
96      0095 1  |     CHECK_PATHNAME            | Sets value state by scanning pathname desc
97      0096 1  |     DBG$NPATHDESC_TO_CS      | Translates a p.n. desc to a c.s.
98      0097 1  |     SCOPE_SCANNER             | Kernel scanner for parsing scopes
99      0098 1  |     DBG$NPARSE_SCOPE_LIST;    | Parses scopes list
100     0099 1  |
101     0100 1  |
102     0101 1  | INCLUDE FILES:
103     0102 1  |
104     0103 1  |
105     0104 1  | REQUIRE 'SRC$:DBGPROLOG.REQ';
106     0238 1  |
107     0239 1  |
108     0240 1  | MACROS:
109     0241 1  |
110     0242 1  |
111     0243 1  | MACRO ! These are parsing and semantic action macros
112     0244 1  |
113     0245 1  |
114     0246 1  | ! ADVANCE updates the input descriptor to reflect the ingestion
115     0247 1  | ! of the current lexical string. In addition, a copy of the lexical string
116     0248 1  | ! descriptor is made.
117     0249 1  |
118     M 0250 1  | ADVANCE =
119     M 0251 1  |     BEGIN
120     M 0252 1  |
121     M 0253 1  |     ch$move (8, lex_string_desc, last_token_desc);
122     M 0254 1  |
123     M 0255 1  |     input_desc [dsc$w_length] = .input_desc [dsc$w_length] -
124     M 0256 1  |                                     (.lex_string_desc [dsc$a_pointer] -
125     M 0257 1  |                                     .input_desc [dsc$a_pointer] +
126     M 0258 1  |                                     .lex_string_desc [dsc$w_length]);
127     M 0259 1  |     input_desc [dsc$a_pointer] = .lex_string_desc [dsc$a_pointer] +
128     M 0260 1  |                                     .lex_string_desc [dsc$w_length];
129     M 0261 1  |
130     M 0262 1  |     last_token = .token;
131     M 0263 1  |
```



```

: 132      0264 1      END %,
: 133      0265 1
: 134      0266 1
: 135      0267 1      ! GET_TOKEN calls the lexical scanner for a token
: 136      0268 1
: 137      M 0269 1      GET_TOKEN =
: 138      M 0270 1      BEGIN
: 139      M 0271 1
: 140      M 0272 1      BIND
: 141      M 0273 1      ROUTINE LEXICAL_SCANNER = .token_scanner_addr; ! Lexical analyzer
: 142      M 0274 1      lexical_scanner (.input_desc, lex_string_desc, token);
: 143      M 0275 1
: 144      M 0276 1
: 145      M 0277 1
: 146      M 0278 1      ! Check for an integer with a length GTR than 9. If this is the case,
: 147      M 0279 1      ! change token to invalid.
: 148      M 0280 1
: 149      M 0281 1      IF .token EQL dbg$k_tok_int
: 150      M 0282 1      THEN
: 151      M 0283 1          IF .lex_string_desc [dsc$w_length] GTR 9
: 152      M 0284 1          THEN
: 153      M 0285 1              token = dbg$k_tok_inval;
: 154      M 0286 1
: 155      0287 1      END %,
: 156      0288 1
: 157      0289 1
: 158      0290 1      ! SAVE extracts and saves the values of the present input descriptor
: 159      0291 1
: 160      M 0292 1      SAVE (LEN, PTR) =
: 161      M 0293 1      BEGIN
: 162      M 0294 1
: 163      M 0295 1          len = .input_desc [dsc$w_length];
: 164      M 0296 1          ptr = .input_desc [dsc$a_pointer];
: 165      M 0297 1
: 166      0298 1      END %,
: 167      0299 1
: 168      0300 1
: 169      0301 1      ! RESTORE sets the present input descriptor values to the ones supplied
: 170      0302 1
: 171      M 0303 1      RESTORE (LEN, PTR) =
: 172      M 0304 1      BEGIN
: 173      M 0305 1
: 174      M 0306 1          input_desc [dsc$w_length] = len;
: 175      M 0307 1          input_desc [dsc$a_pointer] = ptr;
: 176      M 0308 1
: 177      0309 1      END %,
: 178      0310 1
: 179      0311 1
: 180      0312 1      ! ADD_TO_LIST adds a counted string to the name list. If there is no room
: 181      0313 1      ! to add the name, a string truncation message is issued. The count fields
: 182      0314 1      ! of the pathname vector are updated.
: 183      0315 1
: 184      M 0316 1      ADD_TO_LIST (COUNTED_STRING) =
: 185      M 0317 1      BEGIN
: 186      M 0318 1
: 187      M 0319 1          IF .name_index GEQ dbg$k_max_pathname
: 188      M 0320 1          THEN
```



```
189 M 0321 1 SIGNAL (dbg$_pathlong) ! No return
190 M 0322 1 ELSE
191 M 0323 1 BEGIN
192 M 0324 1 name_vect [.name_index] = counted_string;
193 M 0325 1 name_index = .name_index + 1;
194 M 0326 1 END;
195 M 0327 1
196 M 0328 1
197 M 0329 1 ! Update the count fields
198 M 0330 1 !
199 M 0331 1 pathname_desc [pth$b_totcnt] = .pathname_desc [pth$b_totcnt] + 1;
200 M 0332 1 pathname_desc [pth$b_pathcnt] = .pathname_desc [pth$b_totcnt];
201 M 0333 1
202 M 0334 1 END %,
203 M 0335 1
204 M 0336 1
205 M 0337 1 ! ADD_ID adds a non_null name to the name vector. The contents of the lexical
206 M 0338 1 string buffer is copied into a new buffer.
207 M 0339 1
208 M 0340 1 ADD_ID =
209 M 0341 1 BEGIN
210 M 0342 1 LOCAL
211 M 0343 1 NAME_STRING : REF VECTOR [,BYTE]; ! Vector for counted string
212 M 0344 1
213 M 0345 1 ! Determine how large a buffer is needed and allocate it.
214 M 0346 1 !
215 M 0347 1 name_string = dbg$get_tempmem
216 M 0348 1 ((.lex_string_desc [dsc$w_length] / %UPVAL) + 1);
217 M 0349 1
218 M 0350 1
219 M 0351 1 ! Copy the buffer pointed to by the lexical string into the name buffer.
220 M 0352 1 !
221 M 0353 1 ch$move (.lex_string_desc [dsc$w_length],
222 M 0354 1 .lex_string_desc [dsc$a_pointer],
223 M 0355 1 name_string[1]);
224 M 0356 1
225 M 0357 1 name_string[0] = .lex_string_desc [dsc$w_length];
226 M 0358 1
227 M 0359 1
228 M 0360 1 ! Add the buffer to the name vector
229 M 0361 1 !
230 M 0362 1 add_to_list (.name_string);
231 M 0363 1
232 M 0364 1 END %,
233 M 0365 1
234 M 0366 1
235 M 0367 1
236 M 0368 1 ! ADD_INVOCATION_NUMBER attaches an invocation number to the last name added
237 M 0369 1 ! to the name list. The invocation number augmentation is set.
238 M 0370 1
239 M 0371 1 ADD_INVOCATION_NUMBER =
240 M 0372 1 BEGIN
241 M 0373 1 LOCAL
242 M 0374 1 POINTER, ! Temporary pointer
243 M 0375 1 NUMBER_DESC : dbg$stg_desc, ! Descriptor for number
244 M 0376 1 NUM_BUF : REF VECTOR [,BYTE], ! Number buffer
245 M 0377 1
```



```

246 M 0378 1      NUMBER;                                ! Translated number
247 M 0379 1
248 M 0380 1      augmentations [invocation_found] = true;
249 M 0381 1
250 M 0382 1
251 M 0383 1      ! A copy of the present lexical string descriptor must be made which
252 M 0384 1      ! contains a terminating character (<CR>).
253 M 0385 1
254 M 0386 1      number_desc [dsc$w_length] = .lex_string_desc [dsc$w_length] + 1;
255 M 0387 1
256 M 0388 1
257 M 0389 1      ! Allocate storage for the number string and terminator
258 M 0390 1
259 M 0391 1      num_buf = dbg$get_tempmem((.number_desc [dsc$w_length] / %UPVAL) + 1);
260 M 0392 1
261 M 0393 1
262 M 0394 1      ! Copy over the number string and place the terminator
263 M 0395 1
264 M 0396 1      pointer = ch$move (.lex_string_desc [dsc$w_length],
265 M 0397 1      ! .lex_string_desc [dsc$a_pointer],
266 M 0398 1      ! .num_buf);
267 M 0399 1      ch$move (1, UPLIT BYTE (dbg$k_car_return), .pointer);
268 M 0400 1      number_desc [dsc$a_pointer] = .num_buf;
269 M 0401 1
270 M 0402 1
271 M 0403 1      ! The descriptor has been set up. Now convert the number.
272 M 0404 1
273 M 0405 1      IF NOT dbg$nsave_decimal_integer (number_desc, number, dummy)
274 M 0406 1      THEN
275 M 0407 1          RETURN sts$k_severe;
276 M 0408 1
277 M 0409 1
278 M 0410 1      ! Store the invocation number and the index
279 M 0411 1
280 M 0412 1      pathname_desc [pth$b_locinvoc] = .name_index;
281 M 0413 1      pathname_desc [pth$l_invocnum] = .number;
282 M 0414 1
283 M 0415 1      END %,
284 M 0416 1
285 M 0417 1
286 M 0418 1      ! ADD_NULL_ID adds a null name string to the name vector to represent a
287 M 0419 1      ! global reference or numeric scope. The null string is always the first name.
288 M 0420 1
289 M 0421 1      ADD_NULL_ID =
290 M 0422 1      BEGIN
291 M 0423 1
292 M 0424 1      ! Write in the address of the null name into the first name spot
293 M 0425 1
294 M 0426 1      name_vect [0] = null_string;
295 M 0427 1      pathname_desc [pth$b_totcnt] = .pathname_desc [pth$b_totcnt] + 1;
296 M 0428 1      pathname_desc [pth$b_pathcnt] = .pathname_desc [pth$b_totcnt];
297 M 0429 1      name_index = 1;
298 M 0430 1
299 M 0431 1      END %,
300 M 0432 1
301 M 0433 1
302 M 0434 1      ! ADD_GLOBAL_ID inserts the null string into the name list, followed by the
```



```
.. 303      0435 1      ! present id (in the lexical string)
304      0436 1
305      M 0437 1      ADD_GLOBAL_ID =
306      M 0438 1      BEGIN
307      M 0439 1
308      M 0440 1      add_null_id;
309      M 0441 1      add_id;
310      M 0442 1
311      0443 1      END %,
312      0444 1
313      0445 1
314      0446 1      ! ADD_NUMERIC_SCOPE places the null string into the name list and sets up an
315      0447 1      ! invocation number for it (corresponding to the numeric scope). The invocations
316      0448 1      ! augmentation is set by add_invocation_number.
317      0449 1
318      M 0450 1      ADD_NUMERIC_SCOPE =
319      M 0451 1      BEGIN
320      M 0452 1
321      M 0453 1      add_null_id;
322      M 0454 1      add_invocation_number;
323      M 0455 1
324      0456 1      END %,
325      0457 1
326      0458 1
327      0459 1      ! ADD_LINE inserts a '%LINE' followed by the line number into the name list.
328      0460 1      ! LINE augmentations are set.
329      0461 1
330      M 0462 1      ADD_LINE =
331      M 0463 1      BEGIN
332      M 0464 1
333      M 0465 1      LOCAL
334      M 0466 1      LINE_ITEM : REF VECTOR [,BYTE];
335      M 0467 1
336      M 0468 1      augmentations [line_found] = true;
337      M 0469 1      augmentations [line_pending] = false;
338      M 0470 1
339      M 0471 1
340      M 0472 1      ! Get storage for the string
341      M 0473 1      !
342      M 0474 1      line_item = dbg$get_tempmem(((.number_buffer [0] + 6) / %UPVAL) + 1);
343      M 0475 1
344      M 0476 1
345      M 0477 1      ! Copy in the 'LINE'
346      M 0478 1      !
347      M 0479 1      ch$move (6, UPLIT BYTE ('%LINE '), line_item [1]);
348      M 0480 1
349      M 0481 1
350      M 0482 1      ! Copy over the number
351      M 0483 1      !
352      M 0484 1      ch$move (.number_buffer [0], number_buffer [1], line_item [7]);
353      M 0485 1
354      M 0486 1
355      M 0487 1      ! Fill in the count
356      M 0488 1      !
357      M 0489 1      line_item [0] = 6 + .number_buffer [0];
358      M 0490 1
359      M 0491 1
```

```

360      M 0492 1      ! Add the string to the name list
361      M 0493 1      !
362      M 0494 1      add_to_list (.line_item);
363      M 0495 1      !
364      M 0496 1      END %,
365      M 0497 1      !
366      M 0498 1      !
367      M 0499 1      ! ADD_LABEL adds '%LABEL' followed by the label number to the name list and
368      M 0500 1      ! sets the label found augmentation.
369      M 0501 1      !
370      M 0502 1      ADD_LABEL =
371      M 0503 1      -BEGIN
372      M 0504 1      !
373      M 0505 1      LOCAL
374      M 0506 1      LABEL_ITEM : REF VECTOR [,BYTE];
375      M 0507 1      !
376      M 0508 1      augmentations [label_found] = true;
377      M 0509 1      augmentations [label_pending] = false;
378      M 0510 1      !
379      M 0511 1      ! Get storage for the string
380      M 0512 1      !
381      M 0513 1      label_item = dbg$get_tempmem(((.number_buffer [0] + 7) / %UPVAL) + 1);
382      M 0514 1      !
383      M 0515 1      !
384      M 0516 1      ! Copy in the 'LABEL'
385      M 0517 1      !
386      M 0518 1      !
387      M 0519 1      ch$move (7, UPLIT BYTE ('%LABEL '), label_item [1]);
388      M 0520 1      !
389      M 0521 1      ! Copy over the number
390      M 0522 1      !
391      M 0523 1      !
392      M 0524 1      ch$move (.number_buffer [0], number_buffer [1], label_item [8]);
393      M 0525 1      !
394      M 0526 1      !
395      M 0527 1      ! Fill in the count
396      M 0528 1      !
397      M 0529 1      label_item [0] = 7 + .number_buffer [0];
398      M 0530 1      !
399      M 0531 1      !
400      M 0532 1      ! Add the string to the name list
401      M 0533 1      !
402      M 0534 1      add_to_list (.label_item);
403      M 0535 1      !
404      M 0536 1      END %,
405      M 0537 1      !
406      M 0538 1      !
407      M 0539 1      ! ADD_TO_L_NUMBER adds pieces of a line or label number to the number buffer.
408      M 0540 1      ! An augmentation is used to check if this is the first part of the number or
409      M 0541 1      ! a continuation.
410      M 0542 1      !
411      M 0543 1      ADD_TO_L_NUMBER =
412      M 0544 1      -BEGIN
413      M 0545 1      !
414      M 0546 1      LOCAL
415      M 0547 1      NUMBER_DESC : dbg$stg_desc,
416      M 0548 1      TEMP : REF VECTOR [,BYTE];
```



```

: 417      M 0549 1
: 418      M 0550 1
: 419      M 0551 1
: 420      M 0552 1
: 421      M 0553 1
: 422      M 0554 1
: 423      M 0555 1
: 424      M 0556 1
: 425      M 0557 1
: 426      M 0558 1
: 427      M 0559 1
: 428      M 0560 1
: 429      M 0561 1
: 430      M 0562 1
: 431      M 0563 1
: 432      M 0564 1
: 433      M 0565 1
: 434      M 0566 1
: 435      M 0567 1
: 436      M 0568 1
: 437      M 0569 1
: 438      M 0570 1
: 439      M 0571 1
: 440      M 0572 1
: 441      M 0573 1
: 442      M 0574 1
: 443      M 0575 1
: 444      M 0576 1
: 445      M 0577 1
: 446      M 0578 1
: 447      M 0579 1
: 448      M 0580 1
: 449      M 0581 1
: 450      M 0582 1
: 451      M 0583 1
: 452      M 0584 1
: 453      M 0585 1
: 454      M 0586 1
: 455      M 0587 1
: 456      M 0588 1
: 457      M 0589 1
: 458      M 0590 1
: 459      M 0591 1
: 460      M 0592 1
: 461      M 0593 1
: 462      M 0594 1
: 463      M 0595 1
: 464      M 0596 1
: 465      M 0597 1
: 466      M 0598 1
: 467      M 0599 1
: 468      M 0600 1
: 469      M 0601 1
: 470      M 0602 1
: 471      M 0603 1
: 472      M 0604 1
: 473      M 0605 1

number_desc [dsc$a_pointer] = .lex_string_desc [dsc$a_pointer];
number_desc [dsc$w_length] = .lex_string_desc [dsc$w_length];

! Delete leading '0's
!
WHILE .number_desc [dsc$w_length] GTR 1
DO
    BEGIN
        IF ch$rchar (.number_desc [dsc$a_pointer]) NEQ '0'
        THEN
            EXITLOOP;

        number_desc [dsc$w_length] = .number_desc [dsc$w_length] - 1;
        number_desc [dsc$a_pointer] = .number_desc [dsc$a_pointer] + 1;
    END;
    ! End of loop

! Check for new number or continuation
!
IF .augmentations [l_number_started]
THEN
    BEGIN
        ! Add the new number to what we already have
        !
        temp = .number_buffer;
        number_buffer = dbg$get_tempmem
            ((?temp [0] + .number_desc [dsc$w_length]) / %UPVAL) + 1);

        ! concatenate the old string with the new
        !
        ch$move (.temp [0], temp [1], number_buffer [1]);
        ch$move (.number_desc [dsc$w_length],
            .number_desc [dsc$a_pointer],
            number_buffer [.temp [0] + 1]);

        number_buffer [0] = .temp [0] + .number_desc [dsc$w_length];
    END
ELSE
    BEGIN
        ! Start a new number buffer
        !
        augmentations [l_number_started] = true;
        number_buffer = dbg$get_tempmem
            ((?.number_desc [dsc$w_length] / %UPVAL) + 1));

        ch$move (.number_desc [dsc$w_length],
            .number_desc [dsc$a_pointer],
            number_buffer [1]);

        number_buffer [0] = .number_desc [dsc$w_length];
    END;
```



```

: 474      M 0606 1
: 475      0607 1      END %;
: 476      0608 1
: 477      0609 1
: 478      0610 1
: 479      0611 1      EQUATED SYMBOLS:
: 480      0612 1
: 481      0613 1
: 482      0614 1      LITERAL
: 483      0615 1
: 484      0616 1      ! These are augmentation literals
: 485      0617 1
: 486      0618 1      LINE_PENDING          = 0,
: 487      0619 1      LINE_FOUND            = 1,
: 488      0620 1      LABEL_PENDING         = 2,
: 489      0621 1      LABEL_FOUND          = 3,
: 490      0622 1      INVOCATION_FOUND      = 4,
: 491      0623 1      L_NUMBER_STARTED     = 5,
: 492      0624 1      TERMINAL_PENDING     = 6,
: 493      0625 1      TERMINAL_STATE       = 7;
: 494      0626 1
: 495      0627 1
: 496      0628 1      OWN STORAGE:
: 497      0629 1
: 498      0630 1
: 499      0631 1      OWN
500      0632 1      LAST_TOKEN_DESC : dbg$stg_desc,
501      0633 1
502      0634 1      LAST_TOKEN,
503      0635 1      DUMMY,
504      0636 1      INPUT_DESC      : REF dbg$stg_desc,
505      0637 1      PATHNAME_DESC  : REF pth$pathname,
506      0638 1      NAME_VECT     : REF VECTOR,
507      0639 1      NAME_INDEX,
508      0640 1      VALUE_STATE,
509      0641 1      NUMBER_BUFFER  : REF VECTOR [BYTE],
510      0642 1      AUGMENTATIONS : BITVECTOR [8],
511      0643 1      TOKEN,
512      0644 1      TOKEN_SCANNER_ADDR,
513      0645 1      LEX_STRING_DESC : dbg$stg_desc;
514      0646 1
515      0647 1      BIND
516      0648 1      NULL_STRING      = UPLIT BYTE (0);
517      0649 1
518      0650 1
519      0651 1      EXTERNAL REFERENCES:
520      0652 1
521      0653 1
522      0654 1      EXTERNAL ROUTINE
523      0655 1      SYSSFAO          : ADDRESSING_MODE (ABSOLUTE),
524      0656 1      DBG$NNEXT_WORD,
525      0657 1      DBG$NSYNTAX_ERROR,
526      0658 1      DBG$NMATCH,
527      0659 1      DBG$NOUT_INFO,
528      0660 1      DBG$NMAKE_ARG_VECT,
529      0661 1      DBG$GET_TEMP_MEM,
: 530      0662 1      DBG$NSAVE_DECIMAL_INTEGER;

! Copy of last lex string desc
! accepted during parsing
! Last token found
! Dummy variable
! Input string descriptor
! Path name descriptor
! Name vector for pathname descriptor
! Index into name vector
! Return state value
! Buffer for l number
! Augmentation vector
! Lexical token
! Address of lexical scanner
! Descriptor of string for token

! Null string

! System service
! Returns next word of input
! Constructs a syntax error
! Matches input to counted strings
! Outputs an informational message
! Constructs a message argument vector
! Gets listed dynamic storage
! Converts ascii to integer
```



DBGNPNP  
V04-000

L 1  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 11  
(2)

:	531	0663	1			
:	532	0664	1	EXTERNAL		
:	533	0665	1	DBG\$GB_LANGUAGE: BYTE	:	Current language
:	534	0666	1	DBG\$GL_ORIG_COMMAND_PTR	:	Pointer to original command string
:	535	0667	1	DBG\$GL_UPCASE_COMMAND_PTR: VECTOR[2]	:	
:	536	0668	1		:	Pointers to start and end
:	537	0669	1		:	of current command string
:	538	0670	1			



```
540 0671 1 GLOBAL ROUTINE DBG$NPATHNAME_PARSER (INPUT, SCANNER, PATHNAME, VALUE, LAST_DESC) =
541 0672 1
542 0673 1
543 0674 1
544 0675 1
545 0676 1
546 0677 1
547 0678 1
548 0679 1
549 0680 1
550 0681 1
551 0682 1
552 0683 1
553 0684 1
554 0685 1
555 0686 1
556 0687 1
557 0688 1
558 0689 1
559 0690 1
560 0691 1
561 0692 1
562 0693 1
563 0694 1
564 0695 1
565 0696 1
566 0697 1
567 0698 1
568 0699 1
569 0700 1
570 0701 1
571 0702 1
572 0703 1
573 0704 1
574 0705 1
575 0706 1
576 0707 1
577 0708 1
578 0709 1
579 0710 1
580 0711 1
581 0712 1
582 0713 1
583 0714 1
584 0715 1
585 0716 1
586 0717 1
587 0718 1
588 0719 1
589 0720 1
590 0721 1
591 0722 1
592 0723 1
593 0724 1
594 0725 1
595 0726 1
596 0727 1
```

GLOBAL ROUTINE DBG\$NPATHNAME\_PARSER (INPUT, SCANNER, PATHNAME, VALUE, LAST\_DESC) =

++  
FUNCTIONAL DESCRIPTION:

Top level parse network for DEBUG pathname parsing. This network accepts valid DEBUG pathnames and constructs a partial pathname descriptor. Upon return, the caller of this routine must analyze the pathname descriptor in conjunction with the return value, and complete the pathname descriptor.

This routine will not terminate the collection of a pathname until a null or invalid token has been returned by the scanner routine, or an invalid pathname construct has been encountered. This means that the collected pathname may include part or all of a data item reference.

This routine expects to have the address of a language specific lexical analyzer routine passed to it. This lexical analyzer supplies tokens to the parser. The tokens recognized are:

dbg\$k_tok_null	- end of input
dbg\$k_tok_line	- '%LINE'
dbg\$k_tok_label	- '%LABEL'
dbg\$k_tok_bs	- '\' (back slash)
dbg\$k_tok_id	- language specific symbolic identifier
dbg\$k_tok_int	- unsigned integer
dbg\$k_tok_dot	- '.'
dbg\$k_tok_reg	- '%register'
dbg\$k_tok_qname	- '%NAME'
dbg\$k_tok_inval	- any other string

In conjunction with a token, the scanner routine returns a lexical string which contains the ascii characters associated with the token. Note that integers are not translated into binary values by the scanner.

The pathname parser assumes the responsibility of updating the input string to reflect the acceptance of a lexical string corresponding to a token.

Upon success or failure, the input string descriptor is updated to reflect the point at which processing stopped. That is, the dsc\$a\_pointer field contains the address of the first character not accepted.

FORMAL PARAMETERS:

INPUT	- The address of a VAX standard string descriptor representing the input string
-------	---



597	0728	1	SCANNER	- The address of a language specific lexical analyzer
598	0729	1		
599	0730	1	PATHNAME	- The address of a longword to contain the address of a pathname descriptor
600	0731	1		
601	0732	1		
602	0733	1	VALUE	- The address of a longword to contain an unsigned integer encoding of the type of pathname collected:
603	0734	1		
604	0735	1		
605	0736	1		
606	0737	1		dbg\$sk_line - pathname describes %LINE entity, NOT a data item
607	0738	1		
608	0739	1		
609	0740	1		dbg\$sk_label - pathname describes %LABEL entity, NOT a data item
610	0741	1		
611	0742	1		
612	0743	1		dbg\$sk_pn_reg - pathname qualified register (not supported yet), or unqualified register reference (supported). In both cases, the register name is NOT written into the pathname descriptor, but is left as the first token in the input buffer. This means that the pathname descriptor for an unqualified register will have an item count of 0.
613	0744	1		
614	0745	1		
615	0746	1		
616	0747	1		
617	0748	1		
618	0749	1		
619	0750	1		
620	0751	1		
621	0752	1		
622	0753	1		dbg\$sk_pn - pathname may describe a data or lexical entity
623	0754	1		
624	0755	1		
625	0756	1	LAST_DESC	- The address of a longword to contain the address of a standard string descriptor. This descriptor is a copy of the last lexical string descriptor accepted during parsing
626	0757	1		
627	0758	1		
628	0759	1		
629	0760	1		
630	0761	1	[SCOPE_FLAG]	- Optional parameter. If supplied, and if true, then accept global and numeric scopes as well as regular pathnames.
631	0762	1		
632	0763	1		
633	0764	1		
634	0765	1		
635	0766	1	IMPLICIT INPUTS:	
636	0767	1		
637	0768	1	NONE	
638	0769	1		
639	0770	1	IMPLICIT OUTPUTS:	
640	0771	1		
641	0772	1	NONE	
642	0773	1		
643	0774	1	ROUTINE VALUE:	
644	0775	1		
645	0776	1	An unsigned integer longword completion code	
646	0777	1		
647	0778	1	COMPLETION CODES:	
648	0779	1		
649	0780	1	ST\$K_SUCCESS	- Success. Some flavor of pathname returned
650	0781	1		
651	0782	1	ST\$K_SEVERE	- Failure. Syntax error encountered. VALUE parameter not defined. Input descriptor returned to original state.
652	0783	1		
653	0784	1		



```

: 654      0785 1  ! SIDE EFFECTS:
: 655      0786 1  !
: 656      0787 1  !       The input string descriptor is updated to reflect one character beyond the
: 657      0788 1  !       last character accepted.
: 658      0789 1  ! --
: 659      0790 1  !
: 660      0791 2  BEGIN
: 661      0792 2  !
: 662      0793 2  BUILTIN
: 663      0794 2  ACTUALCOUNT,
: 664      0795 2  ACTUALPARAMETER;
: 665      0796 2  !
: 666      0797 2  LOCAL
: 667      0798 2  SCOPE_FLAG;
: 668      0799 2  ! Optional parameter value
: 669      0800 2  ! Set the scope flag
: 670      0801 2  !
: 671      0802 2  scope_flag = (IF actualcount () GIR 5 THEN actualparameter (6) ELSE 0);
: 672      0803 2  !
: 673      0804 2  !
: 674      0805 2  ! All this routine does is to initialize the control variables and call the
: 675      0806 2  ! the real parse network.
: 676      0807 2  !
: 677      0808 2  input_desc = .input;
: 678      0809 2  token_scanner_addr = .scanner;
: 679      0810 2  !
: 680      0811 2  lex_string_desc [dsc$b_class] = dsc$k_class_s;
: 681      0812 2  lex_string_desc [dsc$b_dtype] = dsc$k_dtype_t;
: 682      0813 2  lex_string_desc [dsc$w_length] = 0;
: 683      0814 2  lex_string_desc [dsc$a_pointer] = 0;
: 684      0815 2  !
: 685      0816 2  last_token_desc [dsc$a_pointer] = .input_desc [dsc$a_pointer];
: 686      0817 2  last_token_desc [dsc$w_length] = .input_desc [dsc$w_length];
: 687      0818 2  !
: 688      0819 2  !
: 689      0820 2  ! Obtain storage for the pathname descriptor and line up the name vector
: 690      0821 2  !
: 691      0822 2  pathname_desc = dbg$get_tempmem(dbg$k_pathname_size);
: 692      0823 2  name_vect = pathname_desc [pth$a_pathvector];
: 693      0824 2  name_index = 0;
: 694      0825 2  !
: 695      0826 2  !
: 696      0827 2  ! Initialize the fields of the pathname descriptor.
: 697      0828 2  !
: 698      0829 2  pathname_desc [pth$b_totcnt] = 0;
: 699      0830 2  pathname_desc [pth$b_locinvoc] = 0;
: 700      0831 2  pathname_desc [pth$l_invocnum] = 0;
: 701      0832 2  !
: 702      0833 2  !
: 703      0834 2  ! Initialize the augmentation vector and set the value state
: 704      0835 2  !
: 705      0836 2  augmentations [line_pending] = false;
: 706      0837 2  augmentations [line_found] = false;
: 707      0838 2  augmentations [label_pending] = false;
: 708      0839 2  augmentations [label_found] = false;
: 709      0840 2  augmentations [invocation_found] = false;
: 710      0841 2  augmentations [l_number_started] = false;
```



```
: 711      0842  2      augmentations [terminal_pending] = false;
: 712      0843  2      augmentations [terminal_state]   = false;
: 713      0844  2
: 714      0845  2      value_state = -1;
: 715      0846  2
: 716      0847  2
: 717      0848  2      ! Variables are initialized. Try to do the parsing.
: 718      0849  2      ! Check for scope acceptance.
: 719      0850  2
: 720      0851  2      IF .scope_flag
: 721      0852  2      THEN
: 722      0853  3          BEGIN
: 723      0854  3              IF short_scope ()
: 724      0855  3              THEN
: 725      0856  4                  BEGIN
: 726      0857  4                      .pathname = .pathname_desc;
: 727      0858  4                      RETURN sts$k_success;
: 728      0859  4                  END
: 729      0860  3              ELSE
: 730      0861  4                  BEGIN
: 731      0862  4                      IF parse_pathname ()
: 732      0863  4                      THEN
: 733      0864  5                          BEGIN
: 734      0865  5                              .pathname = .pathname_desc;
: 735      0866  5                              RETURN sts$k_success;
: 736      0867  5                          END
: 737      0868  4                      ELSE
: 738      0869  4                          RETURN sts$k_severe;
: 739      0870  3                  END;
: 740      0871  3              END
: 741      0872  2      ELSE
: 742      0873  3          BEGIN
: 743      0874  3              IF NOT parse_pathname () THEN RETURN sts$k_severe;
: 744      0875  2          END;
: 745      0876  2
: 746      0877  2
: 747      0878  2      ! Set the value state
: 748      0879  2      !
: 749      0880  2      check_pathname ();
: 750      0881  2
: 751      0882  2
: 752      0883  2      ! Return all the expected values.
: 753      0884  2      !
: 754      0885  2      .pathname = .pathname_desc;
: 755      0886  2      .value = .value_state;
: 756      0887  2      .last_desc = last_token_desc;
: 757      0888  2
: 758      0889  2      RETURN sts$k_success;
: 759      0890  2
: 760      0891  1      END;                                !End of DBG$NPATHNAME_PARSER
```

```
.TITLE  DBGNPNP
.IDENT  \V04-000\
.PSECT  DBG$PLIT,NOWRT, SHR, PIC,0
```



00 00000 P.AAA: .BYTE 0  
.PSECT DBG\$OWN,NOEXE, PIC,2

00000 LAST\_TOKEN\_DESC:  
.BLKB 12  
0000C LAST\_TOKEN:  
.BLKB 4  
00010 DUMMY: .BLKB 4  
00014 INPUT\_DESC:  
.BLKB 4  
00018 PATHNAME\_DESC:  
.BLKB 4  
0001C NAME\_VECT:  
.BLKB 4  
00020 NAME\_INDEX:  
.BLKB 4  
00024 VALUE\_STATE:  
.BLKB 4  
00028 NUMBER\_BUFFER:  
.BLKB 4  
0002C AUGMENTATIONS:  
.BLKB 1  
0002D .BLKB 3  
00030 TOKEN: .BLKB 4  
00034 TOKEN\_SCANNER\_ADDR:  
.BLKB 4  
00038 LEX\_STRING\_DESC:  
.BLKB 12

NULL\_STRING= P.AAA  
.EXTRN SYSS\$FAO, DBG\$NNEXT\_WORD  
.EXTRN DBG\$NSYNTAX\_ERROR  
.EXTRN DBG\$NMATCH, DBG\$NOUT\_INFO  
.EXTRN DBG\$NMAKE\_ARG\_VECT  
.EXTRN DBG\$GET\_TEMP\_MEM  
.EXTRN DBG\$NSAVE\_DECIMAL\_INTEGER  
.EXTRN DBG\$GB\_LANGUAGE  
.EXTRN DBG\$GL\_ORIG\_COMMAND\_PTR  
.EXTRN DBG\$GL\_UPCASE\_COMMAND\_PTR

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

			000C 00000	.ENTRY	DBG\$NPATHNAME_PARSER, Save R2,R3	: 0671
53	00000000'	EF	9E 00002	MOVAB	PATHNAME_DESC, R3	: 0802
05		6C	91 00009	CMPB	(AP), #5	: 0808
		06	1B 0000C	BLEQU	1\$	: 0809
52	18	AC	D0 0000E	MOVL	24(AP), SCOPE_FLAG	: 0813
		02	11 00012	BRB	2\$	: 0814
		52	D4 00014	CLRL	SCOPE_FLAG	: 0816
FC	A3	AC	D0 00016	MOVL	INPUT, INPUT_DESC	: 0817
1C	A3	AC	D0 0001B	MOVL	SCANNER, TOKEN_SCANNER_ADDR	: 0818
20	A3 010E0000	8F	D0 00020	MOVL	#17694720, LEX_STRING_DESC	: 0819
		A3	D4 00028	CLRL	LEX_STRING_DESC+4	: 0820
	50	A3	D0 0002B	MOVL	INPUT_DESC, R0	: 0821
EC	A3	A0	D0 0002F	MOVL	4(R0), LAST_TOKEN_DESC+4	: 0822
E8	A3	60	B0 00034	MOVW	(R0), LAST_TOKEN_DESC	: 0823



00000000G	00	34	DD	00038	PUSHL	#52	: 0822
	63	01	FB	0003A	CALLS	#1, DBG\$GET_TEMP MEM	: 0823
04	A3	50	D0	00041	MOVL	R0, PATHNAME_DESC	: 0824
		08	A0	9E 00044	MOVAB	8(R0), NAME_VECT	: 0829
		08	A3	D4 00049	CLRL	NAME_INDEX	: 0830
			60	94 0004C	CLRB	(R0)	: 0831
		02	A0	94 0004E	CLRB	2(R0)	: 0843
		04	A0	D4 00051	CLRL	4(R0)	: 0845
		14	A3	94 00054	CLRB	AUGMENTATIONS	: 0851
0C	A3	01	CE	00057	MNEGL	#1, VALUE_STATE	: 0854
	16	52	E9	0005B	BLBC	SCOPE_FLAG, 4\$	: 0862
0000V	CF	00	FB	0005E	CALLS	#0, SHORT_SCOPE	: 0865
	08	50	E8	00063	BLBS	R0, 3\$	: 0869
0000V	CF	00	FB	00066	CALLS	#0, PARSE_PATHNAME	: 0874
	0E	50	E9	0006B	BLBC	R0, 5\$	: 0880
0C	BC	63	D0	0006E 3\$:	MOVL	PATHNAME_DESC, @PATHNAME	: 0885
		1F	11	00072	BRB	7\$	: 0886
0000V	CF	00	FB	00074 4\$:	CALLS	#0, PARSE_PATHNAME	: 0887
	04	50	E8	00079	BLBS	R0, 6\$	: 0889
	50	04	D0	0007C 5\$:	MOVL	#4, R0	: 0891
			04	0007F	RET		
0000V	CF	00	FB	00080 6\$:	CALLS	#0, CHECK_PATHNAME	: 0892
0C	BC	63	D0	00085	MOVL	PATHNAME_DESC, @PATHNAME	: 0893
10	BC	0C	A3	D0 00089	MOVL	VALUE_STATE, @VALUE	: 0894
14	BC	E8	A3	9E 0008E	MOVAB	LAST_TOKEN_DESC, @LAST_DESC	: 0895
	50	01	D0	00093 7\$:	MOVL	#1, R0	: 0896
		04	00096	RET			

; Routine Size: 151 bytes, Routine Base: DBG\$CODE + 0000

; 761 0892 1



```

: 763      0893 1 ROUTINE PARSE_PATHNAME =
: 764      0894 1
: 765      0895 1 !++
: 766      0896 1 FUNCTIONAL DESCRIPTION:
: 767      0897 1
: 768      0898 1     This routine recognizes legal DEBUG pathnames. All special cases are
: 769      0899 1     trapped first, then the routine goes into a loop to accept the remaining
: 770      0900 1     elements of the pathname. Augmentations are used to assure the the '%L'
: 771      0901 1     constructs appear only one time, as well as to check the validity of
: 772      0902 1     invocation numbers and numeric pathnames.
: 773      0903 1
: 774      0904 1 FORMAL PARAMETERS:
: 775      0905 1
: 776      0906 1     NONE
: 777      0907 1
: 778      0908 1 IMPLICIT INPUTS:
: 779      0909 1
: 780      0910 1     Numerous MODULE LEVEL OWN'ed variables.
: 781      0911 1
: 782      0912 1 IMPLICIT OUTPUTS:
: 783      0913 1
: 784      0914 1     The pathname descriptor is constructed for valid pathname references.
: 785      0915 1
: 786      0916 1 ROUTINE VALUE:
: 787      0917 1
: 788      0918 1     An unsigned integer longword completion code
: 789      0919 1
: 790      0920 1 COMPLETION CODES:
: 791      0921 1
: 792      0922 1     STS$K_SUCCESS    (1)    - Success. Pathname constructed.
: 793      0923 1
: 794      0924 1     STS$K_SEVERE    (4)    - Failure. Illegal pathname.
: 795      0925 1
: 796      0926 1 SIDE EFFECTS:
: 797      0927 1
: 798      0928 1     NONE
: 799      0929 1
: 800      0930 1 --
```



```

: 802      0931 2 BEGIN
: 803      0932
: 804      0933 ! Get the first token and check for all the legal pathname beginnings
: 805      0934 !
: 806      0935 get_token;
: 807      0936
: 808      0937 CASE .token FROM dbg$tok_lowest TO dbg$tok_highest
: 809      0938 OF
: 810      0939 SET
: 811      0940
: 812      0941 [dbg$tok_line] :
: 813      0942 IF NOT first_line () THEN RETURN sts$k_severe;
: 814      0943
: 815      0944 [dbg$tok_label] :
: 816      0945 IF NOT first_label () THEN RETURN sts$k_severe;
: 817      0946
: 818      0947 [dbg$tok_bs] : ! Looking for a global reference
: 819      0948 IF NOT global_item () THEN RETURN sts$k_severe;
: 820      0949
: 821      0950 [dbg$tok_id] : ! starting with an id
: 822      0951 IF NOT id_item () THEN RETURN sts$k_severe;
: 823      0952
: 824      0953 [dbg$tok_int] : ! Numeric scope
: 825      0954 IF NOT numeric_pathname () THEN RETURN sts$k_severe;
: 826      0955
: 827      0956 [dbg$tok_reg] :
: 828      0957 RETURN sts$k_success;
: 829      0958
: 830      0959 [dbg$tok_qname] :
: 831      0960 IF NOT qname_item () THEN RETURN sts$k_severe;
: 832      0961
: 833      0962 [INRANGE, OTRANGE] : ! Error
: 834      0963 RETURN sts$k_severe;
: 835      0964
: 836      0965 TES;
```



```

: 838      0966      2
: 839      0967
: 840      0968
: 841      0969
: 842      0970
: 843      0971
: 844      0972
: 845      0973
: 846      0974
: 847      0975
: 848      0976
: 849      0977
: 850      0978
: 851      0979
: 852      0980
: 853      0981
: 854      0982
: 855      0983
: 856      0984
: 857      0985
: 858      0986
: 859      0987
: 860      0988
: 861      0989
: 862      0990
: 863      0991
: 864      0992
: 865      0993
: 866      0994
: 867      0995
: 868      0996
: 869      0997
: 870      0998
: 871      0999
: 872      1000
: 873      1001
: 874      1002
: 875      1003
: 876      1004
: 877      1005
: 878      1006
: 879      1007
: 880      1008
: 881      1009
: 882      1010      2

! Loop, collecting the rest of the pathname
get_token;
WHILE .token EQL dbg$k_tok_bs AND NOT .augmentations [terminal_state]
DO
    BEGIN
    ! Check for one more trip through loop
    IF .augmentations [terminal_pending]
    THEN
        augmentations [terminal_state] = true;

    advance;
    get_token;

    CASE .token FROM dbg$k_tok_lowest TO dbg$k_tok_highest
    OF
    SET
        [dbg$k_tok_line] : ! '%LINE'
        IF NOT line_item () THEN RETURN sts$k_severe;

        [dbg$k_tok_label] : ! '%LABEL'
        IF NOT label_item () THEN RETURN sts$k_severe;

        [dbg$k_tok_id] : ! ID found. May have an invocation number.
        IF NOT id_item () THEN RETURN sts$k_severe;

        [dbg$k_tok_int] : ! LINE or LABEL number
        IF NOT integer_item () THEN RETURN sts$k_severe;

        [dbg$k_tok_qname] : ! %NAME 'name'
        IF NOT qname_item () THEN RETURN sts$k_severe;

        [INRANGE, OUTRANGE] : ! Error
        RETURN sts$k_severe;

    TES;

    ! Obtain the next token
    !
    get_token;

    END;          ! End of loop
```



```

: 884      1011 2
: 885      1012
: 886      1013
: 887      1014
: 888      1015
: 889      1016
: 890      1017
: 891      1018
: 892      1019
: 893      1020
: 894      1021
: 895      1022
: 896      1023
: 897      1024
: 898      1025
: 899      1026
: 900      1027
: 901      1028
: 902      1029
: 903      1030
: 904      1031
: 905      1032
: 906      1033 1

! Must end parsing on eol
IF .token NEQ dbg$tok_null
    AND
    .token NEQ dbg$tok_inval
    AND
    .token NEQ dbg$tok_id
    AND
    (.token NEQ dbg$tok_dot OR .last_token NEQ dbg$tok_id)
THEN
    RETURN sts$severe;

! See if a '%LINE' or '%LABEL' has been left dangling
IF .augmentations [line_pending] OR .augmentations [label_pending]
THEN
    RETURN sts$severe;

RETURN sts$success;

END;                ! End of parse_pathname
```

```

                                007C 00000 PARSE_PATHNAME:
                                .WORD Save R2,R3,R4,R5,R6
                                MOVAB TOKEN, R6
                                PUSHAB R6
                                PUSHAB LEX_STRING_DESC
                                PUSHAB INPUT_DESC
                                CALLS #3, @TOKEN_SCANNER_ADDR
                                CMPL TOKEN, #6
                                BNEQ 1$
                                CMPW LEX_STRING_DESC, #9
                                BLEQU 1$
                                MOVL #1, TOKEN
                                CASEL TOKEN, #0, #9
                                .WORD 25$-2$,-
                                25$-2$,-
                                4$-2$,-
                                5$-2$,-
                                6$-2$,-
                                7$-2$,-
                                8$-2$,-
                                25$-2$,-
                                26$-2$,-
                                21$-2$
                                25$
                                BRW #0, FIRST_LINE
                                CALLS #0, FIRST_LABEL
                                BRB 9$
                                CALLS #0, GLOBAL_ITEM
                                BRB 9$

001E      0017      010D      002C      00DF      010D      0025      002F      0111
010D      0033      010D      0025      002F      0111

0000V CF      00F6 31 0003B 3$: BRW
0000V CF      00 00 FB 0003E 4$: CALLS
0000V CF      1A 11 00043 5$: BRB
0000V CF      00 00 FB 00045 5$: CALLS
0000V CF      13 11 0004A 6$: BRB
0000V CF      00 00 FB 0004C 6$: CALLS

                                0893
                                0931
                                0937
                                0963
                                0942
                                0945
                                0948
```



				0C 11 00051	BRB 9\$		
	0000V	CF		00 FB 00053 7\$:	CALLS #0, ID_ITEM		0951
				05 11 00058	BRB 9\$		
	0000V	CF		00 FB 0005A 8\$:	CALLS #0, NUMERIC_PATHNAME		0954
		D9		50 E9 0005F 9\$:	BLBC R0, 3\$		0960
				56 DD 00062 10\$:	PUSHL R6		0965
			08	A6 9F 00064	PUSHAB LEX_STRING_DESC		
			E4	A6 DD 00067	PUSHL INPUT_DESC		
	04	B6		03 FB 0006A	CALLS #3, @TOKEN_SCANNER_ADDR		
		06		66 D1 0006E	CMPL TOKEN, #6		
				09 12 00071	BNEQ 11\$		
		09	08	A6 B1 00073	CMPL LEX_STRING_DESC, #9		
				03 1B 00077	BLEQU 11\$		
		66		01 D0 00079	MOVL #1, TOKEN		
		04		66 D1 0007C 11\$:	CMPL TOKEN, #4		0970
				03 13 0007F	BEQL 13\$		
			008D	31 00081 12\$:	BRW 23\$		
			FC	A6 95 00084 13\$:	TSTB AUGMENTATIONS		
				F8 19 00087	BLSS 12\$		
	05	FC	A6	06 E1 00089	BBC #6, AUGMENTATIONS, 14\$		0975
		FC	A6	80 8F 88 0008E	BISB2 #128, AUGMENTATIONS		0977
		08	A6	03 28 00093 14\$:	MOVCS #8, LEX_STRING_DESC, LAST_TOKEN_DESC		
DO	A6		E4	A6 D0 00099	MOVL INPUT_DESC, R1		
			04	A1 C3 0009D	SUBL3 4(R1), LEX_STRING_DESC+4, R0		
	50	OC	A6	08 A6 3C 000A3	MOVZWL LEX_STRING_DESC, R2		
				52 C0 000A7	ADDL2 R2, R0		
				50 A2 000AA	SUBW2 R0, (R1)		
		04	A1	0C B642 9E 000AD	MOVAB @LEX_STRING_DESC+4[R2], 4(R1)		
		DC	A6	66 D0 000B3	MOVL TOKEN, LAST_TOKEN		
				56 DD 000B7	PUSHL R6		0979
			08	A6 9F 000B9	PUSHAB LEX_STRING_DESC		
				51 DD 000BC	PUSHL R1		
	04	B6		03 FB 000BE	CALLS #3, @TOKEN_SCANNER_ADDR		
		06		66 D1 000C2	CMPL TOKEN, #6		
				09 12 000C5	BNEQ 15\$		
		09	08	A6 B1 000C7	CMPL LEX_STRING_DESC, #9		
				03 1B 000CB	BLEQU 15\$		
		66		01 D0 000CD	MOVL #1, TOKEN		
		00		66 CF 000D0 15\$:	CASEL TOKEN, #0, #9		0982
001D		09		0060 000D4 16\$:	.WORD 25\$-16\$,-		
0060		0016		0060 000DC	25\$-16\$,-		
		002B		0060 000E4	17\$-16\$,-		
		0032			18\$-16\$,-		
					25\$-16\$,-		
					19\$-16\$,-		
					20\$-16\$,-		
					25\$-16\$,-		
					25\$-16\$,-		
					21\$-16\$		
					25\$		1002
	0000V	CF		4A 11 000E8	BRB 25\$		0987
				00 FB 000EA 17\$:	CALLS #0, LINE_ITEM		
				1A 11 000EF	BRB 22\$		
	0000V	CF		00 FB 000F1 18\$:	CALLS #0, LABEL_ITEM		0990
				13 11 000F6	BRB 22\$		
	0000V	CF		00 FB 000F8 19\$:	CALLS #0, ID_ITEM		0993
				0C 11 000FD	BRB 22\$		
	0000V	CF		00 FB 000FF 20\$:	CALLS #0, INTEGER_ITEM		0996



DBGNPNP  
V04-000

K 2  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 23  
(7)

0000V	CF	05	11	00104	BRB	22\$		
	26	00	FB	00106	CALLS	#0, QNAME_ITEM		0999
		50	E9	0010B	BLBC	R0, 25\$		
		FF51	31	0010E	BRW	10\$		
	50	66	D0	00111	MOVL	TOKEN, R0		1014
		15	13	00114	BEQL	24\$		
	01	50	D1	00116	CMPL	R0, #1		1016
		10	13	00119	BEQL	24\$		
	05	50	D1	0011B	CMPL	R0, #5		1018
		0B	13	0011E	BEQL	24\$		
	07	50	D1	00120	CMPL	R0, #7		1020
		0F	12	00123	BNEQ	25\$		
	05	DC	A6	D1 00125	CMPL	LAST_TOKEN, #5		
		09	12	00129	BNEQ	25\$		
04	FC	05	FC	A6 E8 0012B	BLBS	AUGMENTATIONS, 25\$		1027
		A6	02	E1 0012F	BBC	#2, AUGMENTATIONS, 26\$		
		50	04	D0 00134	MOVL	#4, R0		1029
			04	00137	RET			
	50	01	D0	00138	MOVL	#1, R0		1031
		04	0013B	RET				1033

; Routine Size: 316 bytes, Routine Base: DBG\$CODE + 0097

; 907 1034 1



```

: 909      1035 1 ROUTINE FIRST_LINE =
: 910      1036 1
: 911      1037 1 ++
: 912      1038 1 FUNCTIONAL DESCRIPTION:
: 913      1039 1
: 914      1040 1     This routine is called when the pathname begins with '%LINE'. Special
: 915      1041 1     handling is given to the resolution of line numbers vs. numeric pathnames.
: 916      1042 1
: 917      1043 1 FORMAL PARAMETERS:
: 918      1044 1
: 919      1045 1     NONE
: 920      1046 1
: 921      1047 1 IMPLICIT INPUTS:
: 922      1048 1
: 923      1049 1     Augmentations and MODULE OWN'ed variables.
: 924      1050 1
: 925      1051 1 IMPLICIT OUTPUTS:
: 926      1052 1
: 927      1053 1     NONE
: 928      1054 1
: 929      1055 1 ROUTINE VALUE:
: 930      1056 1
: 931      1057 1     An unsigned integer longword completion code
: 932      1058 1
: 933      1059 1 COMPLETION CODES:
: 934      1060 1
: 935      1061 1     STS$K_SUCCESS           - Success. Part or all of pathname parsed.
: 936      1062 1
: 937      1063 1     STS$K_SEVERE           - Failure. Illegal construct encountered.
: 938      1064 1
: 939      1065 1 SIDE EFFECTS:
: 940      1066 1
: 941      1067 1     All or part of the pathname descriptor may be constructed.
: 942      1068 1
: 943      1069 1 --
: 944      1070 2 BEGIN
: 945      1071 2
: 946      1072 2     augmentations [line_pending] = true;
: 947      1073 2     advance;
: 948      1074 2
: 949      1075 2     ! Get the next token. If it is an integer, we are going to have to
: 950      1076 2     ! do some lookahead to see if it is a line number or numeric scope.
: 951      1077 2     !
: 952      1078 2     get_token;
: 953      1079 2
: 954      1080 2     CASE .token FROM dbg$k_tok_lowest TO dbg$k_tok_highest
: 955      1081 2     OF
: 956      1082 2     SET
: 957      1083 2
: 958      1084 2     [dbg$k_tok_bs] :           ! Do nothing
: 959      1085 2     BEGIN
: 960      1086 2     0;
: 961      1087 2     END;
: 962      1088 2
: 963      1089 2     [dbg$k_tok_id] :           ! ID followed by possible invocation number
: 964      1090 2     IF NOT id_item () THEN RETURN sts$k_severe;           ! Save the id and advance
: 965      1091 2
```

```
: 966      1092  2      [dbg$tok_int] : ! Here we must do lookahead to see if we have a line number
: 967      1093      IF NOT line_lookahead () THEN RETURN sts$severe;
: 968      1094
: 969      1095      [INRANGE,OUTRANGE] : ! Error
: 970      1096      RETURN sts$severe;
: 971      1097
: 972      1098      TES;
: 973      1099
: 974      1100      RETURN sts$success;
: 975      1101
: 976      1102  1      END;                      ! End of FIRST_LINE
```

```
007C 00000 FIRST_LINE:
DO  A6      FC  A6      56 00000000' EF 9E 00002 .WORD Save R2,R3,R4,R5,R6
      08  A6      01 01 88 00009 MOVAB TOKEN, R6
      51      E4  A6      08 08 28 0000D BISB2 #1, AUGMENTATIONS
      50      0C  A6      04 04 A6 D0 00013 MOVC3 #8, LEX_STRING_DESC, LAST_TOKEN_DESC
      61      08  A6      08 08 A6 C3 00017 MOVL INPUT_DESC, R1
      04  A1      52 04 A6 3C 0001D SUBL3 4(R1), LEX_STRING_DESC+4, R0
      DC  A6      50 08 A6 3C 0001D MOVZWL LEX_STRING_DESC, R2
      0C B642 50 52 C0 00021 ADDL2 R2, R0
      66 00 50 A2 00024 SUBW2 R0, (R1)
      04  A1      0C B642 9E 00027 MOVAB @LEX_STRING_DESC+4[R2], 4(R1)
      DC  A6      66 D0 0002D MOVL TOKEN, LAST_TOKEN
      08  A6      56 DD 00031 PUSHL R6
      51 08 08 A6 9F 00033 PUSHAB LEX_STRING_DESC
      04 B6      51 DD 00036 PUSHL R1
      06      03 FB 00038 CALLS #3, @TOKEN_SCANNER_ADDR
      09      66 D1 0003C CMPL TOKEN, #6
      08  A6      09 12 0003F BNEQ 1$
      03 1B 00045 CMPW LEX_STRING_DESC, #9
      01 D0 00047 BLEQU 1$
      66 CF 0004A 1$: MOVL #1, TOKEN
0025 0025 0025 0025 0025 0025 0025 0025 0025 0025 2$: CASEL TOKEN, #0, #9
0025 0016 0025 0025 0025 0025 0025 0025 0025 0025 .WORD 6$-2$,-
                                6$-2$,-
                                6$-2$,-
                                6$-2$,-
                                7$-2$,-
                                3$-2$,-
                                4$-2$,-
                                6$-2$,-
                                6$-2$,-
                                6$-2$,-
                                6$-2$
0000V CF 0F 11 00062 BRB 6$
0000V CF 00 FB 00064 3$: CALLS #0, ID_ITEM
      05 11 00069 BRB 5$
      00 FB 0006B 4$: CALLS #0, LINE_LOOKAHEAD
      50 E8 00070 5$: BLBS R0, 7$
      04 D0 00073 6$: MOVL #4, R0
      04 04 00076 RET
      50 01 D0 00077 7$: MOVL #1, R0
      04 04 0007A RET
```



DBGNPNP  
V04-000

N 2  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 26  
(8)

; Routine Size: 123 bytes,      Routine Base: DBG\$CODE + 01D3

; 977                    1103 1

```

: 979      1104 1 ROUTINE LINE_LOOKAHEAD =
: 980      1105 1
: 981      1106 1 ++
: 982      1107 1 FUNCTIONAL DESCRIPTION:
: 983      1108 1
: 984      1109 1     Distinguishes between line numbers and numeric pathname items when '%LINE'
: 985      1110 1     is encountered first in pathname parsing.
: 986      1111 1
: 987      1112 1     If the numeric pathname item is found, the entire pathname descriptor is
: 988      1113 1     completed.
: 989      1114 1
: 990      1115 1 FORMAL PARAMETERS:
: 991      1116 1
: 992      1117 1     NONE
: 993      1118 1
: 994      1119 1 IMPLICIT INPUTS:
: 995      1120 1
: 996      1121 1     MODULE OWN'ed variables
: 997      1122 1
: 998      1123 1 IMPLICIT OUTPUTS:
: 999      1124 1
: 1000     1125 1     NONE
: 1001     1126 1
: 1002     1127 1 ROUTINE VALUE:
: 1003     1128 1
: 1004     1129 1     An unsigned integer longword completion code
: 1005     1130 1
: 1006     1131 1 COMPLETION CODES:
: 1007     1132 1
: 1008     1133 1     STS$K_SUCCESS           - Success. Valid entire or partial pathname parsed.
: 1009     1134 1
: 1010     1135 1     STS$K_SEVERE           - Failure. Illegal pathname found.
: 1011     1136 1
: 1012     1137 1 SIDE EFFECTS:
: 1013     1138 1
: 1014     1139 1     Part or all of the pathname descriptor may be constructed.
: 1015     1140 1
: 1016     1141 1 --
: 1017     1142 2 BEGIN
: 1018     1143 2 LOCAL
: 1019     1144 2     LENGTH,
: 1020     1145 2     POINTER;
: 1021     1146 2
: 1022     1147 2     augmentations [line_pending] = true;
: 1023     1148 2
: 1024     1149 2     save (length, pointer);
: 1025     1150 2     advance;
: 1026     1151 2     get_token;
: 1027     1152 2
: 1028     1153 2     CASE .token FROM dbg$k_tok_lowest TO dbg$k_tok_highest
: 1029     1154 2     OF
: 1030     1155 2     SET
: 1031     1156 2
: 1032     1157 2     ! We appear to have a properly terminated line number. Note that we
: 1033     1158 2     ! must accept a line number terminated by an ID as valid because the
: 1034     1159 2     ! ID could be "DO" (as in "SET BREAK %LINE 20 DO(.....)").
: 1035     1160 2
```



```
: 1036      1161  2      [dbg$k_tok_null,
: 1037      1162  2      dbg$k_tok_inval,
: 1038      1163  2      dbg$k_tok_id]:
: 1039      1164  2      BEGIN
: 1040      1165  2      restore (.length, .pointer);
: 1041      1166  2      get_token;
: 1042      1167  2
: 1043      1168  2      IF NOT integer_item () THEN RETURN sts$k_severe;
: 1044      1169  2
: 1045      1170  2      END;
: 1046      1171  2      [dbg$k_tok_bs] : ! Lookahead one more time
: 1047      1172  2      BEGIN
: 1048      1173  2      advance;
: 1049      1174  2      get_token;
: 1050      1175  2
: 1051      1176  2      IF .token EQL dbg$k_tok_int
: 1052      1177  2      THEN
: 1053      1178  3      BEGIN
: 1054      1179  4      ! The first integer we found was a numeric scope
: 1055      1180  4      !
: 1056      1181  4      restore (.length, .pointer);
: 1057      1182  4      get_token;
: 1058      1183  4
: 1059      1184  4      IF NOT numeric_pathname () THEN RETURN sts$k_severe;
: 1060      1185  4      END
: 1061      1186  4      ELSE
: 1062      1187  4      BEGIN
: 1063      1188  3      ! The integer was a line number
: 1064      1189  4      !
: 1065      1190  4      restore (.length, .pointer);
: 1066      1191  4      get_token;
: 1067      1192  4
: 1068      1193  4      IF NOT integer_item () THEN RETURN sts$k_severe;
: 1069      1194  4
: 1070      1195  4      END;
: 1071      1196  4      END;
: 1072      1197  4
: 1073      1198  3      [dbg$k_tok_dot] : ! Line number with a dot
: 1074      1199  2      BEGIN
: 1075      1200  2      restore (.length, .pointer);
: 1076      1201  2      get_token;
: 1077      1202  3
: 1078      1203  3      IF NOT integer_item () THEN RETURN sts$k_severe;
: 1079      1204  3      END;
: 1080      1205  3
: 1081      1206  3      [INRANGE,OUTRANGE] : ! Error
: 1082      1207  2      RETURN sts$k_severe;
: 1083      1208  2
: 1084      1209  2      TES;
: 1085      1210  2      RETURN sts$k_success;
: 1086      1211  2
: 1087      1212  2
: 1088      1213  2
: 1089      1214  2
: 1090      1215  2
: 1091      1216  1      END; ! End of LINE_LOOKAHEAD
```



```
03FC 00000 LINE_LOOKAHEAD:
      59 00000000' EF 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9 ; 1104
FC    A9          01 88 00009 MOVAB TOKEN, R9 ; 1147
      56          E4 A9 D0 0000D BISB2 #1, AUGMENTATIONS ; 1149
      58          66 3C 00011 MOVL INPUT_DESC, R6
      57          04 A6 D0 00014 MOVZWL (R6), LENGTH
      A9          08 08 28 00018 MOVL 4(R6), POINTER
OC    A9          04 A6 C3 0001E MOVCL #8, LEX_STRING_DESC, LAST_TOKEN_DESC
      51          08 A9 3C 00024 SUBL3 4(R6), LEX_STRING_DESC+4, R0
      50          51 C0 00028 MOVZWL LEX_STRING_DESC, R1
      66          50 A2 0002B ADDL2 R1, R0
      A6          0C B9 41 9E 0002E SUBW2 R0, (R6)
DC    A9          69 D0 00034 MOVAB @LEX_STRING_DESC+4[R1], 4(R6)
      59          DD 00038 MOVL TOKEN, LAST_TOKEN ; 1150
      08          A9 9F 0003A PUSHL R9
      56          DD 0003D PUSHAB LEX_STRING_DESC
      04          B9 03 FB 0003F PUSHL R6
      06          69 D1 00043 CALLS #3, @TOKEN_SCANNER_ADDR
      09          08 A9 B1 00048 CMPL TOKEN, #6
      69          01 D0 0004E BNEQ 1$
      00          69 CF 00051 CMPW LEX_STRING_DESC, #9
OOCF 00A3          00A3 00055 BLEQU 1$ ; 1153
OOCF 00A3          0017 0005D MOVL #1, TOKEN
OOCF          00CF 00065 CASEL TOKEN, #0, #9
      00B8 31 00069 .WORD 7$-2$,-
      08 28 0006C 3$: BRW 11$ ; 1210
      A9 D0 00072 MOVCL #8, LEX_STRING_DESC, LAST_TOKEN_DESC ; 1173
      04 A0 C3 00076 MOVL INPUT_DESC, R0
      08 A9 3C 0007C SUBL3 4(R0), LEX_STRING_DESC+4, R1
      52          52 C0 00080 MOVZWL LEX_STRING_DESC, R2
      51          51 A2 00083 ADDL2 R2, R1
      60          50 A2 00088 SUBW2 R1, (R0)
      A0          0C B9 42 9E 00086 MOVAB @LEX_STRING_DESC+4[R2], 4(R0)
DC    A9          69 D0 0008C MOVL TOKEN, LAST_TOKEN ; 1174
      59          DD 00090 PUSHL R9
      08          A9 9F 00092 PUSHAB LEX_STRING_DESC
      50          DD 00095 PUSHL R0
      04          B9 03 FB 00097 CALLS #3, @TOKEN_SCANNER_ADDR
      06          69 D1 0009B CMPL TOKEN, #6
      09          08 A9 B1 000A0 BNEQ 4$
      69          03 1B 000A4 CMPW LEX_STRING_DESC, #9
      50          E4 A9 D0 000A9 MOVL 4$ ; 1183
      50          E4 A9 D0 000A9 MOVL #1, TOKEN
      50          E4 A9 D0 000A9 MOVL INPUT_DESC, R0
```



	06	69	D1	000AD	CMPL	TOKEN, #6	:	1177
		27	12	000B0	BNEQ	6\$	:	
	60	58	B0	000B2	MOVW	LENGTH, (R0)	:	1183
04	A0	57	D0	000B5	MOVL	POINTER, 4(R0)	:	
		59	DD	000B9	PUSHL	R9	:	
		08	A9	9F 000BB	PUSHAB	LEX_STRING_DESC	:	
		50	DD	000BE	PUSHL	R0	:	
04	B9	03	FB	000C0	CALLS	#3, @TOKEN_SCANNER_ADDR	:	
	06	69	D1	000C4	CMPL	TOKEN, #6	:	
		09	12	000C7	BNEQ	5\$	:	
	09	08	A9	B1 000C9	CMPW	LEX_STRING_DESC, #9	:	
		03	1B	000CD	BLEQU	5\$	:	
0000V	69	01	D0	000CF	MOVL	#1, TOKEN	:	
	CF	00	FB	000D2	CALLS	#0, NUMERIC_PATHNAME	:	1186
		48	11	000D7	BRB	10\$	:	
	60	58	B0	000D9	MOVW	LENGTH, (R0)	:	1193
04	A0	57	D0	000DC	MOVL	POINTER, 4(R0)	:	
		59	DD	000E0	PUSHL	R9	:	
		08	A9	9F 000E2	PUSHAB	LEX_STRING_DESC	:	
		50	DD	000E5	PUSHL	R0	:	
04	B9	03	FB	000E7	CALLS	#3, @TOKEN_SCANNER_ADDR	:	
	06	69	D1	000EB	CMPL	TOKEN, #6	:	
		2C	12	000EE	BNEQ	9\$	:	
	09	08	A9	B1 000F0	CMPW	LEX_STRING_DESC, #9	:	
		23	1A	000F4	BGTRU	8\$	:	
		24	11	000F6	BRB	9\$	:	1196
	50	E4	A9	D0 000F8	MOVL	INPUT_DESC, R0	:	1203
	60		58	B0 000FC	MOVW	LENGTH, (R0)	:	
04	A0	57	D0	000FF	MOVL	POINTER, 4(R0)	:	
		59	DD	00103	PUSHL	R9	:	
		08	A9	9F 00105	PUSHAB	LEX_STRING_DESC	:	
		50	DD	00108	PUSHL	R0	:	
04	B9	03	FB	0010A	CALLS	#3, @TOKEN_SCANNER_ADDR	:	
	06	69	D1	0010E	CMPL	TOKEN, #6	:	
		09	12	00111	BNEQ	9\$	:	
	09	08	A9	B1 00113	CMPW	LEX_STRING_DESC, #9	:	
		03	1B	00117	BLEQU	9\$	:	
	69	01	D0	00119	MOVL	#1, TOKEN	:	
0000V	CF	00	FB	0011C	CALLS	#0, INTEGER_ITEM	:	1206
	04	50	E8	00121	BLBS	R0, 12\$	:	
	50	04	D0	00124	MOVL	#4, R0	:	
			04	00127	RET		:	
	50	01	D0	00128	MOVL	#1, R0	:	1214
		04	0012B	RET			:	1216

; Routine Size: 300 bytes, Routine Base: DBG\$CODE + 024E

; 1092 1217 1

```
1094 1218 1 ROUTINE FIRST_LABEL =
1095 1219 1
1096 1220 1 ++
1097 1221 1 FUNCTIONAL DESCRIPTION:
1098 1222 1
1099 1223 1     Invoked when the pathname begins with '%LABEL'. Lookahead may be needed to
1100 1224 1     distinguish a numeric pathname item from a label number.
1101 1225 1
1102 1226 1 FORMAL PARAMETERS:
1103 1227 1
1104 1228 1     NONE
1105 1229 1
1106 1230 1 IMPLICIT INPUTS:
1107 1231 1
1108 1232 1     MODULE level OWN'ed variables
1109 1233 1
1110 1234 1 IMPLICIT OUTPUTS:
1111 1235 1
1112 1236 1     NONE
1113 1237 1
1114 1238 1 ROUTINE VALUE:
1115 1239 1
1116 1240 1     An unsigned integer longword completion code
1117 1241 1
1118 1242 1 COMPLETION CODES:
1119 1243 1
1120 1244 1     STSSK_SUCCESS           - Success. Part or all of a valid pathname parsed.
1121 1245 1
1122 1246 1     STSSK_SEVERE           - Failure. Illegal pathname encountered.
1123 1247 1
1124 1248 1 SIDE EFFECTS:
1125 1249 1
1126 1250 1     Part or all of the pahntame descriptor may be constructed
1127 1251 1
1128 1252 1 --
1129 1253 2 BEGIN
1130 1254 2
1131 1255 2     augmentations [label_pending] = true;
1132 1256 2     advance;
1133 1257 2
1134 1258 2
1135 1259 2     ! Get the next token. If it is an integer, we are going to have to
1136 1260 2     ! do some lookahead to see if it is a label number or numeric scope.
1137 1261 2     !
1138 1262 2     get_token;
1139 1263 2
1140 1264 2     CASE token FROM dbg$tok_lowest TO dbg$tok_highest
1141 1265 2     OF
1142 1266 2     SET
1143 1267 2
1144 1268 2     [dbg$tok_bs] :           ! Do nothing
1145 1269 2     0;
1146 1270 2
1147 1271 2     [dbg$tok_id] :           ! ID followed by possible invocation number
1148 1272 2     IF NOT id_item () THEN RETURN sts$severe;
1149 1273 2
1150 1274 2     [dbg$tok_int] : ! Here we must do lookahead to see if we have
```



Address	Disassembly	Comment	Hex	Symbol
0000	MOVAB	Save R2,R3,R4,R5,R6	9E 00002	1218
0001	TOKEN, R6		04 88 00009	1255
0002	BISB2	#4, AUGMENTATIONS	08 28 0000D	
0003	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	D0 00013	
0004	MOVL	INPUT_DESC, R1	A6 00017	
0005	SUBL3	4(R1), LEX_STRING_DESC+4, R0	A1 C3 0001D	
0006	MOVZWL	LEX_STRING_DESC, R2	A6 3C 00021	
0007	ADDL2	R2, R0	52 C0 00024	
0008	SUBW2	R0, (R1)	50 A2 00027	
0009	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	A1 9E 0002D	
000A	MOVL	TOKEN, LAST_TOKEN	A6 D0 00031	
000B	PUSHL	R6	56 DD 00033	1256
000C	PUSHAB	LEX_STRING_DESC	08 A6 9F 00036	
000D	PUSHL	R1	51 DD 00038	
000E	CALLS	#3, @TOKEN_SCANNER_ADDR	03 FB 0003C	
000F	CMPL	TOKEN, #6	66 D1 0003F	
0010	BNEQ	1\$	09 12 00041	
0011	CMPL	LEX_STRING_DESC, #9	08 A6 1B 00045	
0012	BLEQU	1\$	03 D0 00047	
0013	MOVL	#1, TOKEN	01 CF 0004A	1\$:
0014	CASEL	TOKEN, #0, #9	66 0004E	2\$:
0015	.WORD	6\$-2\$,-	0004E	
0016		6\$-2\$,-	00056	
0017		6\$-2\$,-	0005E	
0018		6\$-2\$,-		
0019		6\$-2\$,-		
001A		7\$-2\$,-		
001B		3\$-2\$,-		
001C		4\$-2\$,-		
001D		6\$-2\$,-		
001E		6\$-2\$,-		
001F		6\$-2\$		
0020	BRB	6\$	0F 11 00062	1279
0021	CALLS	#0, ID_ITEM	00 FB 00064	1272
0022	BRB	5\$	05 11 00069	
0023	CALLS	#0, LABEL_LOOKAHEAD	00 FB 0006B	1276
0024	BLBS	R0, 7\$	50 E8 00070	
0025	MOVL	#4, R0	04 D0 00073	
0026	RET		04 00076	
0027	MOVL	#1, R0	01 D0 00077	1283
0028	RET		04 0007A	1285

DBGNPNP  
V04-000

H 3  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 33  
(10)

; Routine Size: 123 bytes,      Routine Base: DBG\$CODE + 037A

; 1162                    1286 1



```
1164 1287 1 ROUTINE LABEL_LOOKAHEAD =
1165 1288 1
1166 1289 1 ++
1167 1290 1 FUNCTIONAL DESCRIPTION:
1168 1291 1
1169 1292 1     Performs lookahead to distinguish a numeric pathname item from a label number.
1170 1293 1
1171 1294 1     If a numeric pathname item is found, the entire pathname will be parsed.
1172 1295 1
1173 1296 1 FORMAL PARAMETERS:
1174 1297 1
1175 1298 1     NONE
1176 1299 1
1177 1300 1 IMPLICIT INPUTS:
1178 1301 1
1179 1302 1     MODULE level OWN'ed variables, including the augmentation vector.
1180 1303 1
1181 1304 1 IMPLICIT OUTPUTS:
1182 1305 1
1183 1306 1     NONE
1184 1307 1
1185 1308 1 ROUTINE VALUE:
1186 1309 1
1187 1310 1     An unsigned integer longword completion code
1188 1311 1
1189 1312 1 COMPLETION CODES:
1190 1313 1
1191 1314 1     STS$K_SUCCESS           - Success. Part or all of a valid pathname parsed.
1192 1315 1
1193 1316 1     STS$K_SEVERE           - Failure. Invalid pathname found.
1194 1317 1
1195 1318 1 SIDE EFFECTS:
1196 1319 1
1197 1320 1     Part or all of the pathname descriptor may be constructed.
1198 1321 1
1199 1322 1 --
1200 1323 2 BEGIN
1201 1324 2
1202 1325 2 LOCAL
1203 1326 2     LENGTH,
1204 1327 2     POINTER;
1205 1328 2
1206 1329 2     augmentations [label_pending] = true;
1207 1330 2     save (length, pointer);
1208 1331 2     advance;
1209 1332 2     get_token;
1210 1333 2
1211 1334 2 CASE .token FROM dbg$k_tok_lowest TO dbg$k_tok_highest
1212 1335 2     OF
1213 1336 2     SET
1214 1337 2
1215 1338 2         [dbg$k_tok_null,
1216 1339 2         dbg$k_tok_inval,
1217 1340 2         dbg$k_tok_id]:
1218 1341 2         BEGIN
1219 1342 2             restore (.length, .pointer);
1220 1343 2             get_token;
```

03FC 00000 LABEL_LOOKAHEAD:			
	59	00000000'	EF 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9
FC	A9		MOVAB TOKEN, R9
	56	E4	BISB2 #4, AUGMENTATIONS
	58		MOVL INPUT_DESC, R6
	57	04	MOVZWL (R6), -LENGTH
08	A9		MOVL 4(R6), POINTER
0C	A9	04	MOVCL3 #8, LEX_STRING_DESC, LAST_TOKEN_DESC
	51	08	SUBL3 4(R6), LEX_STRING_DESC+4, -R0
			MOVZWL LEX_STRING_DESC, R1

1287  
1329  
1330



		50		51	C0	00028		ADDL2	R1, R0		
		66		50	A2	0002B		SUBW2	R0, (R6)		
	04	A6		OC B941	9E	0002E		MOVAB	@LEX_STRING_DESC+4(R1), 4(R6)		
	DC	A9		69	DO	00034		MOVL	TOKEN, LAST_TOKEN		
				59	DD	00038		PUSHL	R9		1331
				08	A9	9F	0003A	PUSHAB	LEX_STRING_DESC		
					56	DD	0003D	PUSHL	R6		
	04	B9			03	FB	0003F	CALLS	#3, @TOKEN_SCANNER_ADDR		
		06			69	D1	00043	CMPL	TOKEN, #6		
					09	12	00046	BNEQ	1\$		
		09		08	A9	B1	00048	CMPW	LEX_STRING_DESC, #9		
					03	1B	0004C	BLEQU	1\$		
		69			01	DO	0004E	MOVL	#1, TOKEN		
		00			69	CF	00051	CASEL	TOKEN, #0, #9		1334
00D3		00D3	09		0017		00055	.WORD	3\$-2\$,-		
00D3	00D3	00D3	00D3		0017		0005D		3\$-2\$,-		
		00D3			00D3		00065		13\$-2\$,-		
									13\$-2\$,-		
									6\$-2\$,-		
									3\$-2\$,-		
									13\$-2\$,-		
									13\$-2\$,-		
									13\$-2\$,-		
									13\$-2\$,-		
									13\$-2\$		
									13\$		1379
		50		E4	A9	DO	0006C	BRW	INPUT_DESC, R0		1342
		60			58	BO	00070	MOVL	LENGTH, (R0)		
	04	A0			57	DO	00073	MOVW	POINTER, 4(R0)		
					59	DD	00077	MOVL	R9		
				08	A9	9F	00079	PUSHL	R9		
					50	DD	0007C	PUSHAB	LEX_STRING_DESC		
					03	FB	0007E	PUSHL	R0		
	04	B9			69	D1	00082	CALLS	#3, @TOKEN_SCANNER_ADDR		
		06			03	13	00085	CMPL	TOKEN, #6		
					0096	31	00087	BEQL	5\$		
		09		08	A9	B1	0008A	BRW	11\$		
					F7	1B	0008E	CMPW	LEX_STRING_DESC, #9		
					008A	31	00090	BLEQU	4\$		
					08	28	00093	BRW	10\$		
DO	A9	08	A9		A9	DO	00099	MOVW	#8, LEX_STRING_DESC, LAST_TOKEN_DESC		1350
	51		50	E4	A9	DO	00099	MOVL	INPUT_DESC, R0		
			52	04	A0	C3	0009D	SUBL3	4(R0), LEX_STRING_DESC+4, R1		
			51	08	A9	3C	000A3	MOVZWL	LEX_STRING_DESC, R2		
			60		52	C0	000A7	ADDL2	R2, R1		
			A0		51	A2	000AA	SUBW2	R1, (R0)		
			DC	OC B942	9E	000AD		MOVAB	@LEX_STRING_DESC+4(R2), 4(R0)		
				69	DO	000B3		MOVL	TOKEN, LAST_TOKEN		
				59	DD	000B7		PUSHL	R9		1351
				08	A9	9F	000B9	PUSHAB	LEX_STRING_DESC		
					50	DD	000BC	PUSHL	R0		
	04	B9			03	FB	000BE	CALLS	#3, @TOKEN_SCANNER_ADDR		
		06			69	D1	000C2	CMPL	TOKEN, #6		
					09	12	000C5	BNEQ	7\$		
		09		08	A9	B1	000C7	CMPW	LEX_STRING_DESC, #9		
					03	1B	000CB	BLEQU	7\$		
		69			01	DO	000CD	MOVL	#1, TOKEN		
		50		E4	A9	DO	000D0	MOVL	INPUT_DESC, R0		1360
		06			69	D1	000D4	CMPL	TOKEN, #6		1354

			27	12	000D7	BNEQ	9\$		
	04	60	58	B0	000D9	MOVW	LENGTH, (R0)		1360
		A0	57	D0	000DC	MOVL	POINTER, 4(R0)		
			59	DD	000E0	PUSHL	R9		
			08	A9	9F 000E2	PUSHAB	LEX_STRING_DESC		
			50	DD	000E5	PUSHL	R0		
	04	B9	03	FB	000E7	CALLS	#3, @TOKEN_SCANNER_ADDR		
		06	69	D1	000EB	CMPL	TOKEN, #6		
			09	12	000EE	BNEQ	8\$		
		09	08	A9	B1 000F0	CMPW	LEX_STRING_DESC, #9		
			03	1B	000F4	BLEQU	8\$		
	0000V	69	01	D0	000F6	MOVL	#1, TOKEN		1363
		CF	00	FB	000F9	CALLS	#0, NUMERIC_PATHNAME		
			25	11	000FE	BRB	12\$		
		60	58	B0	00100	MOVW	LENGTH, (R0)		1370
	04	A0	57	D0	00103	MOVL	POINTER, 4(R0)		
			59	DD	00107	PUSHL	R9		
			08	A9	9F 00109	PUSHAB	LEX_STRING_DESC		
			50	DD	0010C	PUSHL	R0		
	04	B9	03	FB	0010E	CALLS	#3, @TOKEN_SCANNER_ADDR		
		06	69	D1	00112	CMPL	TOKEN, #6		
			09	12	00115	BNEQ	11\$		
		09	08	A9	B1 00117	CMPW	LEX_STRING_DESC, #9		
			03	1B	0011B	BLEQU	11\$		
	0000V	69	01	D0	0011D	MOVL	#1, TOKEN		1373
		CF	00	FB	00120	CALLS	#0, INTEGER_ITEM		
		04	50	E8	00125	BLBS	R0, 14\$		
		50	04	D0	00128	MOVL	#4, R0		
				04	0012B	RET			
		50	01	D0	0012C	MOVL	#1, R0		1383
			04	0012F	RET				1385

; Routine Size: 304 bytes, Routine Base: DBG\$CODE + 03F5

; 1263 1386 1



```
1265 1387 1 ROUTINE GLOBAL_ITEM =
1266 1388 1
1267 1389 1 !++
1268 1390 1 FUNCTIONAL DESCRIPTION:
1269 1391 1
1270 1392 1     Invoked when the pathname begins with '\'. The entire pathname correspodng
1271 1393 1     to the global reference will be parsed.
1272 1394 1
1273 1395 1 FORMAL PARAMETERS:
1274 1396 1
1275 1397 1     NONE
1276 1398 1
1277 1399 1 IMPLICIT INPUTS:
1278 1400 1
1279 1401 1     MODULE level OWN'ed variables.
1280 1402 1
1281 1403 1 IMPLICIT OUTPUTS:
1282 1404 1
1283 1405 1     NONE
1284 1406 1
1285 1407 1 ROUTINE VALUE:
1286 1408 1
1287 1409 1     An unsigned integer longword completion code
1288 1410 1
1289 1411 1 COMPLETION CODES:
1290 1412 1
1291 1413 1     ST$K_SUCCESS           - Success. Global reference parsed.
1292 1414 1
1293 1415 1     ST$K_SEVERE            _ Failure. Invalid pathname detected.
1294 1416 1
1295 1417 1 SIDE EFFECTS:
1296 1418 1
1297 1419 1     All of the pathname descriptor will be constructed for a valid global
1298 1420 1     reference.
1299 1421 1
1300 1422 1 --
1301 1423 2 BEGIN
1302 1424 2
1303 1425 2     advance;
1304 1426 2     get_token;
1305 1427 2
1306 1428 2
1307 1429 2     ! This must be an id or an id followed by an invocation number
1308 1430 2     !
1309 1431 2     IF .token NEQ dbg$tok_id THEN RETURN sts$severe;
1310 1432 2
1311 1433 2     add_global_id;
1312 1434 2     advance;
1313 1435 2     get_token;
1314 1436 2
1315 1437 2     CASE .token FROM dbg$tok_lowest TO dbg$tok_highest
1316 1438 2     OF
1317 1439 2     SET
1318 1440 2
1319 1441 2     ! Success and end.
1320 1442 2     !
1321 1443 2     [dbg$tok_null,
```

```
: 1322      1444      2      dbg$tok_inval,  
: 1323      1445      2      dbg$tok_id]:  
: 1324      1446      2      BEGIN  
: 1325      1447      2      0;  
: 1326      1448      2      END;  
: 1327      1449      2  
: 1328      1450      2      [dbg$tok_int] : ! Invocation number  
: 1329      1451      2      BEGIN  
: 1330      1452      2      add_invocation_number;  
: 1331      1453      2      advance;  
: 1332      1454      2      END;  
: 1333      1455      2  
: 1334      1456      2      [INRANGE,OUTRANGE] :  
: 1335      1457      2      BEGIN  
: 1336      1458      2      RETURN sts$k_severe;  
: 1337      1459      2      END;  
: 1338      1460      2  
: 1339      1461      2      TES;  
: 1340      1462      2  
: 1341      1463      2      augmentations [terminal_state] = true;  
: 1342      1464      2  
: 1343      1465      2      RETURN sts$k_success;  
: 1344      1466      2  
: 1345      1467      1      END;          ! End of GLOBAL_ITEM
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

OD 00001 P.AAB: .BYTE 13

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

01FC 00000 GLOBAL\_ITEM:

				58	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8	: 1387
				57	00000000'	EF	9E	00009	MOVAB	DBG\$GET_TEMPME, R8	
				5E		10	C2	00010	MOVAB	LEX_STRING_DESC, R7	
				67		08	28	00013	SUBL2	#16, SP	
C8	A7			51	DC	A7	D0	00018	MOVAB	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	: 1423
	50	04		A7	04	A1	C3	0001C	MOVL	INPUT_DESC, R1	
				52		67	3C	00022	SUBL3	4(R1), LEX_STRING_DESC+4, R0	
				50		52	C0	00025	MOVZWL	LEX_STRING_DESC, R2	
				61		50	A2	00028	ADDL2	R2, R0	
				A1	04	B742	9E	0002B	SUBW2	R0, (R1)	
				A7	F8	A7	D0	00031	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	
					F8	A7	9F	00036	MOVL	TOKEN, LAST_TOKEN	: 1425
					0082	8F	BB	00039	PUSHAB	TOKEN	
				FC	B7	03	FB	0003D	PUSHR	#*M<R1,R7>	
				06	F8	A7	D1	00041	CALLS	#3, @TOKEN_SCANNER_ADDR	
				09		09	12	00045	CMPL	TOKEN, #6	
						67	B1	00047	BNEQ	1\$	
						04	1B	0004A	CMPW	LEX_STRING_DESC, #9	
				F8	A7	01	D0	0004C	BLEQU	1\$	
				05	F8	A7	D1	00050	MOVL	#1, TOKEN	: 1431
									CMPL	TOKEN, #5	



PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------



		50		04	C6	00116	DIVL2	#4, R0	
			01	A0	9F	00119	PUSHAB	1(R0)	
		68		01	FB	0011C	CALLS	#1, DBG\$GET_TEMP MEM	
		56		50	D0	0011F	MOVL	R0, NUM_BUF	
66	04	B7		67	28	00122	MOVC3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -	
								(NUM_BUF)	
		63	00000000'	EF	90	00127	MOVB	P.AAB, (POINTER)	
	08	AE		56	D0	0012E	MOVL	NUM_BUF, NUMBER_DESC+4	
				D8	A7	9F	PUSHAB	DUMMY	
				04	AE	9F	PUSHAB	NUMBER	
				0C	AE	9F	PUSHAB	NUMBER_DESC	
00000000G		00		03	FB	0013B	CALLS	#3, DBG\$NSAVE_DECIMAL_INTEGER	
		04		50	E8	00142	BLBS	R0, 9\$	
		50		04	D0	00145	MOVL	#4, R0	
				04	00148		RET		
		50		E0	A7	D0	00149	9\$: MOVL	PATHNAME_DESC, R0
	02	A0		E8	A7	90	0014D	MOVB	NAME_INDEX, 2(R0)
	04	A0			6E	D0	00152	MOVL	NUMBER, 4(R0)
C8	A7	67		08	28	00156	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1452
		50		DC	A7	D0	0015B	MOVL	INPUT_DESC, R0
51	04	A7		04	A0	C3	0015F	SUBL3	4(R0), LEX_STRING_DESC+4, R1
		52			67	3C	00165	MOVZWL	LEX_STRING_DESC, R2
		51			52	C0	00168	ADDL2	R2, R1
		60			51	A2	0016B	SUBW2	R1, (R0)
	04	A0	04	B742	9E	0016E	MOVAB	@LEX_STRING_DESC+4[R2], 4(R0)	
	D4	A7	F8	A7	D0	00174	MOVL	TOKEN, LAST_TOKEN	
	F4	A7	80	8F	88	00179	BISB2	#128, AUGMENTATIONS	1463
		50		01	D0	0017E	MOVL	#1, R0	1465
				04	00181		RET		1467

; Routine Size: 386 bytes, Routine Base: DBG\$CODE + 0525

; 1346 1468 1



```
1348 1469 1 ROUTINE NUMERIC_PATHNAME =
1349 1470 1
1350 1471 1 ++
1351 1472 1 FUNCTIONAL DESCRIPTION:
1352 1473 1
1353 1474 1     Parse the entire pathname when a numeric pathname item is encountered at
1354 1475 1     the start of a pathname.
1355 1476 1
1356 1477 1
1357 1478 1 FORMAL PARAMETERS:
1358 1479 1
1359 1480 1     NONE
1360 1481 1
1361 1482 1 IMPLICIT INPUTS:
1362 1483 1
1363 1484 1     MODULE level OWN'ed variables.
1364 1485 1
1365 1486 1 IMPLICIT OUTPUTS:
1366 1487 1
1367 1488 1     NONE
1368 1489 1
1369 1490 1 ROUTINE VALUE:
1370 1491 1
1371 1492 1     An unsigned integer longword completion code
1372 1493 1
1373 1494 1 COMPLETION CODES:
1374 1495 1
1375 1496 1     STS$K_SUCCESS           - Success. Valid numeric pathname parsed.
1376 1497 1
1377 1498 1     STS$K_SEVERE           - Failure. Invalid pathname found.
1378 1499 1
1379 1500 1 SIDE EFFECTS:
1380 1501 1
1381 1502 1     The entire pathname descriptor for a valid numeric pathname is constructed.
1382 1503 1
1383 1504 1 --
1384 1505 2 BEGIN
1385 1506 2
1386 1507 2     add_numeric_scope;
1387 1508 2     advance;
1388 1509 2     get_token;
1389 1510 2
1390 1511 2
1391 1512 2     ! Looking for backslash
1392 1513 2
1393 1514 2     IF .token NEQ dbg$tok_bs THEN RETURN sts$severe;
1394 1515 2
1395 1516 2     advance;
1396 1517 2     get_token;
1397 1518 2
1398 1519 2
1399 1520 2     ! The data item or '%line', '%label' must immediately follow the numeric scope
1400 1521 2
1401 1522 2     CASE .token FROM dbg$tok_lowest TO dbg$tok_highest
1402 1523 2         OF
1403 1524 2         SET
1404 1525 2
```

```
: 1405      1526      2      [dbg$tok_line] : ! %line
: 1406      1527      2      IF NOT line_item () THEN RETURN sts$k_severe;
: 1407      1528      2
: 1408      1529      2      [dbg$tok_label] : ! '%LABEL'
: 1409      1530      2      IF NOT label_item () THEN RETURN sts$k_severe;
: 1410      1531      2
: 1411      1532      2      [dbg$tok_id] : ! Data reference
: 1412      1533      2      IF NOT id_item () THEN RETURN sts$k_severe;
: 1413      1534      2
: 1414      1535      2      [dbg$tok_int] : ! Possible line or label number
: 1415      1536      2      IF NOT integer_item () THEN RETURN sts$k_severe;
: 1416      1537      2
: 1417      1538      2      [INRANGE, OTRANGE] : ! Error
: 1418      1539      2      RETURN sts$k_severe;
: 1419      1540      2
: 1420      1541      2      TES;
: 1421      1542      2
: 1422      1543      2      augmentations [terminal_state] = true;
: 1423      1544      2
: 1424      1545      2      RETURN sts$k_success;
: 1425      1546      2
: 1426      1547      1      END;          ! End of NUMERIC_PATHNAME
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

OD 00002 P.AAC: .BYTE 13

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

00FC 00000 NUMERIC\_PATHNAME:

		57	00000000'	EF	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7	: 1469
		5E		10	C2	00009	MOVAB	LEX_STRING_DESC, R7	
	E4	B7	00000000'	EF	9E	0000C	SUBL2	#16, SP	
		50	E0	A7	D0	00014	MOVAB	NULL STRING, @NAME_VECT	: 1505
				60	96	00018	MOVL	PATHNAME_DESC, R0	
				01	90	0001A	INCB	(R0)	
	01	A0		01	D0	0001E	MOVB	(R0), 1(R0)	
	E8	A7		10	88	00022	MOVL	#1, NAME_INDEX	
	F4	A7		01	A1	00026	BISB2	#16, AUGMENTATIONS	
04	AE	67		04	AE	3C	ADDW3	#1, LEX_STRING_DESC, NUMBER_DESC	
		50	04	04	C6	0002F	MOVZWL	NUMBER_DESC, R0	
		50	01	A0	9F	00032	DIVL2	#4, R0	
				01	FB	00035	PUSHAB	1(R0)	
	00000000G	00		50	D0	0003C	CALLS	#1, DBG\$GET_TEMPMEM	
		56		67	28	0003F	MOVL	R0, NUM_BUF	
66	04	B7					MOV3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -	
								(NUM_BUF)	
		63	00000000'	EF	90	00044	MOVB	P.AAC, (POINTER)	
	08	AE		56	D0	0004B	MOVL	NUM_BUF, NUMBER_DESC+4	
				A7	9F	0004F	PUSHAB	DUMMY	
			D8	AE	9F	00052	PUSHAB	NUMBER	
			04	AE	9F	00055	PUSHAB	NUMBER_DESC	
			0C	03	FB	00058	CALLS	#3, DBG\$NSAVE_DECIMAL_INTEGER	
	00000000G	00							



		03		50	E8	0005F	BLBS	R0, 1\$		
				00C5	31	00062	BRW	10\$		
		02	50	E0	A7	D0	00065	1\$:	MOVL	PATHNAME_DESC, R0
		04	A0	E8	A7	90	00069	MOVB	NAME_INDEX, 2(R0)	
C8	A7		67		6E	D0	0006E	MOVL	NUMBER, 4(R0)	
			51	DC	08	28	00072	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1507
	50	04	A7	04	A7	D0	00077	MOVL	INPUT_DESC, R1	
			52		A1	C3	0007B	SUBL3	4(R1), LEX_STRING_DESC+4, R0	
			50		67	3C	00081	MOVZWL	LEX_STRING_DESC, R2	
			61		52	C0	00084	ADDL2	R2, R0	
		04	A1		50	A2	00087	SUBW2	R0, (R1)	
		D4	A7	04	B742	9E	0008A	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	
				F8	A7	D0	00090	MOVL	TOKEN, LAST_TOKEN	
				F8	A7	9F	00095	PUSHAB	TOKEN	1508
				0082	8F	BB	00098	PUSHR	#*M<R1,R7>	
		FC	B7		03	FB	0009C	CALLS	#3, @TOKEN_SCANNER_ADDR	
			06	F8	A7	D1	000A0	CMPL	TOKEN, #6	
			09		09	12	000A4	BNEQ	2\$	
					67	B1	000A6	CMPW	LEX_STRING_DESC, #9	
		F8	A7		04	1B	000A9	BLEQU	2\$	
			04	F8	01	D0	000AB	MOVL	#1, TOKEN	1514
					A7	D1	000AF	CMPL	TOKEN, #4	
C8	A7		67		75	12	000B3	BNEQ	10\$	
			51	DC	08	28	000B5	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	
	50	04	A7	04	A7	D0	000BA	MOVL	INPUT_DESC, R1	
			52		A1	C3	000BE	SUBL3	4(R1), LEX_STRING_DESC+4, R0	
			50		67	3C	000C4	MOVZWL	LEX_STRING_DESC, R2	
			61		52	C0	000C7	ADDL2	R2, R0	
		04	A1		50	A2	000CA	SUBW2	R0, (R1)	
		D4	A7	04	B742	9E	000CD	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	
				F8	A7	D0	000D3	MOVL	TOKEN, LAST_TOKEN	
				F8	A7	9F	000D8	PUSHAB	TOKEN	1516
				0082	8F	BB	000DB	PUSHR	#*M<R1,R7>	
		FC	B7		03	FB	000DF	CALLS	#3, @TOKEN_SCANNER_ADDR	
			06	F8	A7	D1	000E3	CMPL	TOKEN, #6	
			09		09	12	000E7	BNEQ	3\$	
					67	B1	000E9	CMPW	LEX_STRING_DESC, #9	
		F8	A7		04	1B	000EC	BLEQU	3\$	
	09		00		01	D0	000EE	MOVL	#1, TOKEN	
001D	0016		0033	F8	A7	CF	000F2	CASEL	TOKEN, #0, #9	1522
0033	002B		0024		0033		000F7	.WORD	10\$-4\$,-	
			0033		0033		000FF		10\$-4\$,-	
					0033		00107		5\$-4\$,-	
									6\$-4\$,-	
									10\$-4\$,-	
									7\$-4\$,-	
									8\$-4\$,-	
									10\$-4\$,-	
									10\$-4\$,-	
									10\$-4\$,-	
									10\$-4\$	
									10\$	1539
0000V	CF			1D	11	0010B	BRB			
				00	FB	0010D	CALLS	#0, LINE_ITEM		1527
				13	11	00112	BRB	9\$		
0000V	CF			00	FB	00114	CALLS	#0, LABEL_ITEM		1530
				0C	11	00119	BRB	9\$		
0000V	CF			00	FB	0011B	CALLS	#0, ID_ITEM		1533
				05	11	00120	BRB	9\$		

16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

Page 45  
(13)

PC	OP	OP2	OP3	OP4	OP5	OP6	OP7	OP8	OP9	OP10	OP11	OP12	OP13	OP14	OP15	OP16	OP17	OP18	OP19	OP20	OP21	OP22	OP23	OP24	OP25	OP26	OP27	OP28	OP29	OP30	OP31	OP32	OP33	OP34	OP35	OP36	OP37	OP38	OP39	OP40	OP41	OP42	OP43	OP44	OP45	OP46	OP47	OP48	OP49	OP50	OP51	OP52	OP53	OP54	OP55	OP56	OP57	OP58	OP59	OP60	OP61	OP62	OP63	OP64	OP65	OP66	OP67	OP68	OP69	OP70	OP71	OP72	OP73	OP74	OP75	OP76	OP77	OP78	OP79	OP80	OP81	OP82	OP83	OP84	OP85	OP86	OP87	OP88	OP89	OP90	OP91	OP92	OP93	OP94	OP95	OP96	OP97	OP98	OP99	OP100	OP101	OP102	OP103	OP104	OP105	OP106	OP107	OP108	OP109	OP110	OP111	OP112	OP113	OP114	OP115	OP116	OP117	OP118	OP119	OP120	OP121	OP122	OP123	OP124	OP125	OP126	OP127	OP128	OP129	OP130	OP131	OP132	OP133	OP134	OP135	OP136	OP137	OP138	OP139	OP140	OP141	OP142	OP143	OP144	OP145	OP146	OP147	OP148	OP149	OP150	OP151	OP152	OP153	OP154	OP155	OP156	OP157	OP158	OP159	OP160	OP161	OP162	OP163	OP164	OP165	OP166	OP167	OP168	OP169	OP170	OP171	OP172	OP173	OP174	OP175	OP176	OP177	OP178	OP179	OP180	OP181	OP182	OP183	OP184	OP185	OP186	OP187	OP188	OP189	OP190	OP191	OP192	OP193	OP194	OP195	OP196	OP197	OP198	OP199	OP200	OP201	OP202	OP203	OP204	OP205	OP206	OP207	OP208	OP209	OP210	OP211	OP212	OP213	OP214	OP215	OP216	OP217	OP218	OP219	OP220	OP221	OP222	OP223	OP224	OP225	OP226	OP227	OP228	OP229	OP230	OP231	OP232	OP233	OP234	OP235	OP236	OP237	OP238	OP239	OP240	OP241	OP242	OP243	OP244	OP245	OP246	OP247	OP248	OP249	OP250	OP251	OP252	OP253	OP254	OP255	OP256	OP257	OP258	OP259	OP260	OP261	OP262	OP263	OP264	OP265	OP266	OP267	OP268	OP269	OP270	OP271	OP272	OP273	OP274	OP275	OP276	OP277	OP278	OP279	OP280	OP281	OP282	OP283	OP284	OP285	OP286	OP287	OP288	OP289	OP290	OP291	OP292	OP293	OP294	OP295	OP296	OP297	OP298	OP299	OP300	OP301	OP302	OP303	OP304	OP305	OP306	OP307	OP308	OP309	OP310	OP311	OP312	OP313	OP314	OP315	OP316	OP317	OP318	OP319	OP320	OP321	OP322	OP323	OP324	OP325	OP326	OP327	OP328	OP329	OP330	OP331	OP332	OP333	OP334	OP335	OP336	OP337	OP338	OP339	OP340	OP341	OP342	OP343	OP344	OP345	OP346	OP347	OP348	OP349	OP350	OP351	OP352	OP353	OP354	OP355	OP356	OP357	OP358	OP359	OP360	OP361	OP362	OP363	OP364	OP365	OP366	OP367	OP368	OP369	OP370	OP371	OP372	OP373	OP374	OP375	OP376	OP377	OP378	OP379	OP380	OP381	OP382	OP383	OP384	OP385	OP386	OP387	OP388	OP389	OP390	OP391	OP392	OP393	OP394	OP395	OP396	OP397	OP398	OP399	OP400	OP401	OP402	OP403	OP404	OP405	OP406	OP407	OP408	OP409	OP410	OP411	OP412	OP413	OP414	OP415	OP416	OP417	OP418	OP419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------

: 1427 1548 1



```
1429 1549 1 ROUTINE LINE_ITEM =
1430 1550 1
1431 1551 1 ++
1432 1552 1 FUNCTIONAL DESCRIPTION:
1433 1553 1
1434 1554 1     Accepts a '%LINE' line_number construct.
1435 1555 1
1436 1556 1 FORMAL PARAMETERS:
1437 1557 1
1438 1558 1     NONE
1439 1559 1
1440 1560 1 IMPLICIT INPUTS:
1441 1561 1
1442 1562 1     MODULE level OWN'ed variables.
1443 1563 1
1444 1564 1 IMPLICIT OUTPUTS:
1445 1565 1
1446 1566 1     NONE
1447 1567 1
1448 1568 1 ROUTINE VALUE:
1449 1569 1
1450 1570 1     An unsigned integer longword completion code
1451 1571 1
1452 1572 1 COMPLETION CODES:
1453 1573 1
1454 1574 1     STS$K_SUCCESS           - Success. Line item parsed.
1455 1575 1
1456 1576 1     STS$K_SEVERE           - Failure. Invalid line item.
1457 1577 1
1458 1578 1 SIDE EFFECTS:
1459 1579 1
1460 1580 1     The '%LINE' reference is added to the pathname descriptor
1461 1581 1
1462 1582 1 --
1463 1583 2 BEGIN
1464 1584 2
1465 1585 2 ! Check to see if we have already encountered '%LINE' or '%LABEL'
1466 1586 2 !
1467 1587 2 IF .augmentations [line_pending] OR .augmentations [label_pending]
1468 1588 2     OR
1469 1589 2     .augmentations [line_found] OR .augmentations [label_found]
1470 1590 2
1471 1591 2 THEN
1472 1592 2     RETURN sts$k_severe;
1473 1593 2
1474 1594 2     augmentations [line_pending] = true;
1475 1595 2     advance;
1476 1596 2     get_token;
1477 1597 2
1478 1598 2
1479 1599 2 ! Accept the line number
1480 1600 2 !
1481 1601 2 IF .token NEQ dbg$k_tok_int THEN RETURN sts$k_severe;
1482 1602 2
1483 1603 2 IF NOT integer_item () THEN RETURN sts$k_severe;
1484 1604 2
1485 1605 2 RETURN sts$k_success;
```

: 1486  
: 14871606 2  
1607 1

END;

! END of LINE\_ITEM

```
                                007C 00000 LINE_ITEM:
                                .WORD
56 00000000' EF 9E 00002   MOVAB   Save R2,R3,R4,R5,R6      : 1549
5E 66 E8 00009   BLBS   AUGMENTATIONS, R6
66 02 E0 0000C   BBS    AUGMENTATIONS, 2$      : 1587
5A 66 01 E0 00010 BBS    #2, AUGMENTATIONS, 2$
56 66 03 E0 00014 BBS    #1, AUGMENTATIONS, 2$      : 1589
52 66 01 88 00018 BBS    #3, AUGMENTATIONS, 2$
66 08 28 0001B   BISB2  #1, AUGMENTATIONS
D4 A6 0C A6 08 28 0001B   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC : 1594
51 E8 A6 D0 00021   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
50 04 A1 C3 00025   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
52 0C A6 3C 0002B   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
61 50 A2 00032   ADDL2  #8, LEX_STRING_DESC, LAST_TOKEN_DESC
04 A1 10 B6 42 9E 00035   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
E0 A6 04 A6 D0 0003B   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC : 1595
04 A6 04 A6 9F 00040   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
0C A6 04 A6 9F 00043   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
08 B6 04 A6 D1 0004C   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
06 0A 12 00050   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
09 0C A6 B1 00052   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
04 A6 04 1B 00056   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
06 01 D0 00058   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
0000V CF 00 FB 00062   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
04 50 E8 00067   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
50 04 D0 0006A   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
50 04 04 0006D   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC
01 D0 0006E   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC : 1605
04 00071   MOVAB   #8, LEX_STRING_DESC, LAST_TOKEN_DESC : 1607
```

; Routine Size: 114 bytes, Routine Base: DBG\$CODE + 07DE

: 1488

1608 1



```
1490 1609 1 ROUTINE LABEL_ITEM =
1491 1610 1
1492 1611 1 ++
1493 1612 1 FUNCTIONAL DESCRIPTION:
1494 1613 1
1495 1614 1     Parses a '%LABEL' item.
1496 1615 1
1497 1616 1 FORMAL PARAMETERS:
1498 1617 1
1499 1618 1     NONE
1500 1619 1
1501 1620 1 IMPLICIT INPUTS:
1502 1621 1
1503 1622 1     The augmentation vector.
1504 1623 1
1505 1624 1 IMPLICIT OUTPUTS:
1506 1625 1
1507 1626 1     NONE
1508 1627 1
1509 1628 1 ROUTINE VALUE:
1510 1629 1
1511 1630 1     An unsigned integer longword completion code
1512 1631 1
1513 1632 1 COMPLETION CODES:
1514 1633 1
1515 1634 1     STS$K_SUCCESS           - Success. Label item parsed.
1516 1635 1
1517 1636 1     STS$K_SEVERE            - Failure. Invalid label item.
1518 1637 1
1519 1638 1 SIDE EFFECTS:
1520 1639 1
1521 1640 1     The label reference is added to the pathname descriptor.
1522 1641 1
1523 1642 1 --
1524 1643 2 BEGIN
1525 1644 2
1526 1645 2 ! See if '%LINE' or '%LABEL' has already been found
1527 1646 2 !
1528 1647 2 IF .augmentations [line_pending] OR .augmentations [label_pending]
1529 1648 2     OR
1530 1649 2     .augmentations [line_found] OR .augmentations [label_found]
1531 1650 2
1532 1651 2 THEN
1533 1652 2     RETURN sts$K_severe;
1534 1653 2
1535 1654 2     augmentations [label_pending] = true;
1536 1655 2     advance;
1537 1656 2     get_token;
1538 1657 2
1539 1658 2
1540 1659 2 ! Accept the label number
1541 1660 2 !
1542 1661 2 IF .token NEQ dbg$K_tok_int THEN RETURN sts$K_severe;
1543 1662 2
1544 1663 2 IF NOT integer_item () THEN RETURN sts$K_severe;
1545 1664 2
1546 1665 2 RETURN sts$K_success;
```

: 1547  
: 15481666 2  
1667 1

END:

! End of LABEL\_ITEM

```
                                007C 00000 LABEL_ITEM:
                                .WORD
56 00000000' EF 9E 00002      MOVAB Save R2,R3,R4,R5,R6      : 1609
5E 66 E8 00009      BLBS AUGMENTATIONS, R6
66 02 E0 0000C      BBS AUGMENTATIONS, 2$      : 1647
5A 66 01 E0 00010      BBS #2, AUGMENTATIONS, 2$
56 66 03 E0 00014      BBS #1, AUGMENTATIONS, 2$      : 1649
52 66 04 88 00018      BISB2 #3, AUGMENTATIONS, 2$
66 08 28 0001B      MOV3 #4, AUGMENTATIONS
D4 A6 0C A6 08 28 0001B      MOV3 #8, LEX_STRING_DESC, LAST_TOKEN_DESC : 1654
51 E8 A6 D0 00021      MOVL INPUT_DESC, R1
50 04 A1 C3 00025      SUBL3 4(R1), LEX_STRING_DESC+4, R0
52 0C A6 3C 0002B      MOVZWL LEX_STRING_DESC, R2
61 50 52 C0 0002F      ADDL2 R2, R0
04 A1 10 B6 42 9E 00035      SUBW2 R0, (R1)
E0 A6 04 A6 D0 0003B      MOVAB @LEX_STRING_DESC+4[R2], 4(R1)
04 A6 9F 00040      MOVL TOKEN, LAST_TOKEN
0C A6 9F 00043      PUSHAB TOKEN      : 1655
51 DD 00046      PUSHAB LEX_STRING_DESC
08 B6 03 FB 00048      PUSHL R1
06 04 A6 D1 0004C      CALLS #3, @TOKEN_SCANNER_ADDR
09 0A 12 00050      CMPL TOKEN, #6
04 A6 B1 00052      BNEQ 1$
06 04 1B 00056      CMPW LEX_STRING_DESC, #9
04 A6 01 D0 00058      BLEQU 1$
06 04 A6 D1 0005C 1$:      MOVL #1, TOKEN      : 1661
0000V CF 00 FB 00062      CMPL TOKEN, #6
04 50 E8 00067      BNEQ 2$
50 04 D0 0006A 2$:      CALLS #0, INTEGER_ITEM      : 1663
50 04 04 0006D      BLBS R0, 3$
01 D0 0006E 3$:      MOVL #4, R0
04 00071      RET      : 1665
                        : 1667
```

; Routine Size: 114 bytes, Routine Base: DBG\$CODE + 0850

: 1549

1668 1



```

: 1551      1669 1 ROUTINE QNAME_ITEM =
: 1552      1670 1
: 1553      1671 1 ++
: 1554      1672 1 FUNCTIONAL DESCRIPTION:
: 1555      1673 1
: 1556      1674 1     Parses an QNAME item.
: 1557      1675 1
: 1558      1676 1 FORMAL PARAMETERS:
: 1559      1677 1
: 1560      1678 1     NONE
: 1561      1679 1
: 1562      1680 1 IMPLICIT INPUTS:
: 1563      1681 1
: 1564      1682 1     The augmentation vector.
: 1565      1683 1
: 1566      1684 1 IMPLICIT OUTPUTS:
: 1567      1685 1
: 1568      1686 1     NONE
: 1569      1687 1
: 1570      1688 1 ROUTINE VALUE:
: 1571      1689 1
: 1572      1690 1     An unsigned integer longword completion code
: 1573      1691 1
: 1574      1692 1 COMPLETION CODES:
: 1575      1693 1
: 1576      1694 1     ST$K_SUCCESS           - Success. Valid QNAME item parsed.
: 1577      1695 1
: 1578      1696 1     ST$K_SEVERE           - Failure. Invalid QNAME item found.
: 1579      1697 1
: 1580      1698 1 SIDE EFFECTS:
: 1581      1699 1
: 1582      1700 1     An ID item is added to the pathname descriptor.
: 1583      1701 1
: 1584      1702 1 --
: 1585      1703 2 BEGIN
: 1586      1704 2 LOCAL
: 1587      1705 2     character : BYTE,
: 1588      1706 2     terminal  : BYTE;
: 1589      1707 2
: 1590      1708 2 BIND ROUTINE lexical_scanner = .token_scanner_addr;
: 1591      1709 2 BIND
: 1592      1710 2     lexeme_length  = lex_string_desc[dsc$w_length] : WORD,
: 1593      1711 2     lexeme_pointer  = lex_string_desc[dsc$a_pointer] : LONG;

```

```
: 1595      1712  2      ! First advance over '%NAME' and any following blanks
: 1596      1713  2      !
: 1597      1714  2      advance;
: 1598      1715  2
: 1599      1716  2      IF .input_desc[dsc$w_length] GTRU 0
: 1600      1717  2      THEN
: 1601      1718  2          BEGIN
: 1602      1719  3              character = ch$rchar(.input_desc[dsc$a_pointer]);
: 1603      1720  3              WHILE .character EQL ' ' AND .input_desc[dsc$w_length] GTRU 0
: 1604      1721  3                  DO
: 1605      1722  4                  BEGIN
: 1606      1723  4                      input_desc[dsc$w_length] = .input_desc[dsc$w_length] - 1;
: 1607      1724  4                      input_desc[dsc$a_pointer] = ch$plus(.input_desc[dsc$a_pointer],1);
: 1608      1725  4                      character = ch$rchar(.input_desc[dsc$a_pointer]);
: 1609      1726  3                  END;
: 1610      1727  2          END;
: 1611      1728  2
: 1612      1729  2      IF .input_desc[dsc$w_length] LEQU 0 THEN RETURN sts$k_severe;
: 1613      1730  2
: 1614      1731  2      IF .character EQL '(' OR .character EQL '"' OR .character EQL dbg$k_quote
: 1615      1732  2      THEN      ! Name is enclosed in delimiters
: 1616      1733  3          BEGIN
: 1617      1734  3              IF .input_desc[dsc$w_length] LEQU 2 THEN RETURN sts$k_severe;
: 1618      1735  3              terminal = (IF .character EQL '(' THEN ')' ELSE .character);
: 1619      1736  3              lexeme_length = 0;
: 1620      1737  3              lexeme_pointer = ch$plus(.input_desc[dsc$a_pointer],1);
: 1621      1738  3              character = ch$rchar(ch$plus(.lexeme_pointer,.lexeme_length));
: 1622      1739  4              WHILE (.character NEQ .terminal)
: 1623      1740  3                  DO
: 1624      1741  4                  BEGIN
: 1625      1742  4                      IF .character EQL dbg$k_car_return THEN RETURN sts$k_severe;
: 1626      1743  4                      lexeme_length = .lexeme_length + 1;
: 1627      1744  4                      IF .lexeme_length + 1 GEQU .input_desc[dsc$w_length]
: 1628      1745  4                          THEN
: 1629      1746  4                              RETURN sts$k_severe;
: 1630      1747  4                      character = ch$rchar(ch$plus(.lexeme_pointer,.lexeme_length));
: 1631      1748  3                  END;
: 1632      1749  3              END
: 1633      1750  2      ELSE
: 1634      1751  3          BEGIN
: 1635      1752  3              lexical_scanner(.input_desc,lex_string_desc,token);
: 1636      1753  3              IF .token NEQ dbg$k_tok_id AND .token NEQ dbg$k_tok_int
: 1637      1754  3                  THEN
: 1638      1755  3                      RETURN sts$k_severe;
: 1639      1756  3              terminal = 0;
: 1640      1757  2          END;
```



```
: 1642      1758 2 token = dbg$tok_id;
: 1643      1759 2 add_id;
: 1644      1760 2 advance;
: 1645      1761 2 IF .terminal NEQ 0
: 1646      1762 2 THEN
: 1647      1763 2     BEGIN
: 1648      1764 2         input_desc[dsc$w_length] = .input_desc[dsc$w_length] - 1;
: 1649      1765 2         input_desc[dsc$a_pointer] = ch$plus(.input_desc[dsc$a_pointer],1);
: 1650      1766 2     END;
: 1651      1767 2
: 1652      1768 2 get_token;
: 1653      1769 2
: 1654      1770 2 ! Check for invocation number
: 1655      1771 2
: 1656      1772 2 IF .token EQL dbg$tok_int
: 1657      1773 2 THEN
: 1658      1774 2     BEGIN ! See if an invocation number has already been found.
: 1659      1775 2         IF .augmentations [invocation_found] THEN RETURN sts$severe;
: 1660      1776 2         add_invocation_number;
: 1661      1777 2         advance;
: 1662      1778 2     END;
: 1663      1779 2
: 1664      1780 2 RETURN sts$success;
: 1665      1781 2
: 1666      1782 1 END;          ! End of QNAME_ITEM
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

OD 00003 P.AAD: .BYTE 13

LEXEME\_LENGTH= LEX\_STRING\_DESC  
LEXEME\_POINTER= LEX\_STRING\_DESC+4

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

03FC 00000 QNAME\_ITEM:

		59	00000000G	00	9E	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9	: 1669
		58	00000000'	EF	9E	00009	MOVAB	DBG\$GET_TEMPMEM, R9	
		5E		10	C2	00010	MOVAB	LEX_STRING_DESC, R8	
		68		08	28	00013	SUBL2	#16, SP	
C8	A8	51	DC	A8	D0	00018	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	: 1711
		52	04	A1	9E	0001C	MOVL	INPUT_DESC, R1	
	50	53		62	C3	00020	MOVAB	4(R1), R2	
		53		68	3C	00025	SUBL3	(R2), LEX_STRING_DESC+4, R0	
		50		53	C0	00028	MOVZWL	LEX_STRING_DESC, R3	
		61		50	A2	0002B	ADDL2	R3, R0	
		62	04	B843	9E	0002E	SUBW2	R0, (R1)	
		62	D4	A8	F8	00033	MOVAB	@LEX_STRING_DESC+4[R3], (R2)	
				61	B5	00038	MOVL	TOKEN, LAST_TOKEN	
				13	13	0003A	TSTW	(R1)	: 1716
		53	00	B2	90	0003C	BEQL	2\$	
		20		53	91	00040	MOVB	@0(R2), CHARACTER	: 1719
				0A	12	00043	CMPB	CHARACTER, #32	: 1720
							BNEQ	2\$	



			61	B5	00045	TSTW	(R1)		
			06	13	00047	BEQL	2\$		
			61	B7	00049	DECW	(R1)		1723
			62	D6	0004B	INCL	(R2)		1724
			ED	11	0004D	BRB	1\$		1725
			61	B5	0004F	TSTW	(R1)		1729
			51	13	00051	BEQL	8\$		
			50	D4	00053	CLRL	R0		1731
		28	53	91	00055	CMPB	CHARACTER, #40		
			04	12	00058	BNEQ	3\$		
			50	D6	0005A	INCL	R0		
			0A	11	0005C	BRB	4\$		
		22	53	91	0005E	CMPB	CHARACTER, #34		
			05	13	00061	BEQL	4\$		
		27	53	91	00063	CMPB	CHARACTER, #39		
			3F	12	00066	BNEQ	9\$		
		02	61	B1	00068	CMPW	(R1), #2		1734
			37	1B	0006B	BLEQU	8\$		
		05	50	E9	0006D	BLBC	R0, 5\$		1735
		50	29	D0	00070	MOVL	#41, R0		
			03	11	00073	BRB	6\$		
		50	53	9A	00075	MOVZBL	CHARACTER, R0		
		57	50	90	00078	MOVB	R0, TERMINAL		
			68	B4	0007B	CLRW	LEXEME_LENGTH		1736
04	A8		62	01	C1	ADDL3	#1, (R2), LEXEME_POINTER		1737
		50	68	3C	00082	MOVZWL	LEXEME_LENGTH, R0		1738
		50	A8	C0	00085	ADDL2	LEXEME_POINTER, R0		
		53	60	90	00089	MOVB	(R0), CHARACTER		
		57	53	91	0008C	CMPB	CHARACTER, TERMINAL		1739
			2F	13	0008F	BEQL	11\$		
		0D	53	91	00091	CMPB	CHARACTER, #13		1742
			0E	13	00094	BEQL	8\$		
			68	B6	00096	INCL	LEXEME_LENGTH		1743
		50	68	3C	00098	MOVZWL	LEXEME_LENGTH, R0		1744
			50	D6	0009B	INCL	R0		
50		61	10	00	ED	CMPZV	#0, #16, (R1), R0		
				DE	1A	BGTRU	7\$		
				00EC	31	BRW	16\$		1746
				F8	A8	PUSHAB	TOKEN		1752
				0102	8F	PUSHR	#^M<R1,R8>		
		FC	B8	03	FB	CALLS	#3, @TOKEN_SCANNER_ADDR		
			05	F8	A8	CPL	TOKEN, #5		1753
					06	BEQL	10\$		
				F8	A8	CPL	TOKEN, #6		
					E6	BNEQ	8\$		
			57	94	000BE	CLRB	TERMINAL		1756
		F8	05	D0	000C0	MOVL	#5, TOKEN		1758
			50	68	3C	MOVZWL	LEX_STRING_DESC, R0		
			50	04	C6	DIVL2	#4, R0		
				01	A0	PUSHAB	1(R0)		
			69	01	FB	CALLS	#1, DBG\$GET TEMPMEM		
			56	50	D0	MOVL	R0, NAME_STRING		
01	A6	04	B8	68	28	MOVC3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -		
							1(NAME_STRING)		
			66	68	90	MOVB	LEX_STRING_DESC, (NAME_STRING)		
			52	A8	D0	MOVL	NAME_INDEX, R2		
			32	52	D1	CPL	R2, #50		



				00000000G 00	00028200	0F 19 000E3	BLSS 12\$		
						8F DD 000E5	PUSHL #164352		
				E4 B842		01 FB 000EB	CALLS #1, LIB\$SIGNAL		
						08 11 000F2	BRB 13\$		
				50	E8	56 D0 000F4	12\$: MOVL NAME_STRING, @NAME_VECT[R2]		
					E0	A8 D6 000F9	INCL NAME_INDEX		
				01 A0		A8 D0 000FC	13\$: MOVL PATHNAME_DESC, R0		
C8	A8			68		60 96 00100	INCB (R0)		
				50	DC	60 90 00102	MOVB (R0), 1(R0)		
51				A8	04	08 28 00106	MOVC3 #8, LEX_STRING_DESC, LAST_TOKEN_DESC		1759
				52		A8 D0 0010B	MOVL INPUT_DESC, R0		
				51		A0 C3 0010F	SUBL3 4(R0), LEX_STRING_DESC+4, R1		
				60		68 3C 00115	MOVZWL LEX_STRING_DESC, R2		
				A0		52 C0 00118	ADDL2 R2, R1		
				D4 A8	04 B842	51 A2 0011B	SUBW2 R1, (R0)		
					F8	9E 0011E	MOVAB @LEX_STRING_DESC+4[R2], 4(R0)		
						A8 D0 00124	MOVL TOKEN, LAST_TOKEN		
						57 95 00129	TSTB TERMINAL		1761
						05 13 0012B	BEQL 14\$		
					04	60 B7 0012D	DECW (R0)		1764
					F8	A0 D6 0012F	INCL 4(R0)		1765
					0101	A8 9F 00132	14\$: PUSHAB TOKEN		1766
				FC B8		8F BB 00135	PUSHR #^M<R0,R8>		
				06	F8	03 FB 00139	CALLS #3, @TOKEN_SCANNER_ADDR		
						A8 D1 0013D	CMPL TOKEN, #6		
				09		09 12 00141	BNEQ 15\$		
						68 B1 00143	CMPL LEX_STRING_DESC, #9		
				04		1B 00146	BLEQU 15\$		
				F8 A8		01 D0 00148	MOVL #1, TOKEN		
				06	F8	A8 D1 0014C	15\$: CMPL TOKEN, #6		1772
						75 12 00150	BNEQ 18\$		
3C				F4 A8		04 E0 00152	BBS #4, AUGMENTATIONS, 16\$		1775
				F4 A8		10 88 00157	BISB2 #16, AUGMENTATIONS		
04	AE			68		01 A1 0015B	ADDW3 #1, LEX_STRING_DESC, NUMBER_DESC		
				50	04	AE 3C 00160	MOVZWL NUMBER_DESC, R0		
				50		04 C6 00164	DIVL2 #4, R0		
					01	A0 9F 00167	PUSHAB 1(R0)		
				69		01 FB 0016A	CALLS #1, DBG\$GET_TEMPMEM		
				56		50 D0 0016D	MOVL R0, NUM_BUF		
66				04 B8		68 28 00170	MOVC3 LEX_STRING_DESC, @LEX_STRING_DESC+4, -		
							(NUM_BUF)		
				63 00000000'		EF 90 00175	MOVB P.AAD, (POINTER)		
				08 AE		56 D0 0017C	MOVL NUM_BUF, NUMBER_DESC+4		
					D8	A8 9F 00180	PUSHAB DUMMY		
					04	AE 9F 00183	PUSHAB NUMBER		
					0C	AE 9F 00186	PUSHAB NUMBER_DESC		
				00000000G 00		03 FB 00189	CALLS #3, DBG\$NSAVE_DECIMAL_INTEGER		
				04		50 E8 00190	BLBS R0, 17\$		
				50		04 D0 00193	16\$: MOVL #4, R0		
						04 00196	RET		
				50	E0	A8 D0 00197	17\$: MOVL PATHNAME_DESC, R0		
				02 A0	E8	A8 90 0019B	MOVB NAME_INDEX, 2(R0)		
C8	A8			04 A0		6E D0 001A0	MOVL NUMBER, 4(R0)		
				68		08 28 001A4	MOVC3 #8, LEX_STRING_DESC, LAST_TOKEN_DESC		1776
				50	DC	A8 D0 001A9	MOVL INPUT_DESC, R0		
51				A8	04	A0 C3 001AD	SUBL3 4(R0), LEX_STRING_DESC+4, R1		
				52		68 3C 001B3	MOVZWL LEX_STRING_DESC, R2		

```

D 5
16-Sep-1984 01:50:44 VAX-11 Bliss-32 V4.0-742
14-Sep-1984 12:17:18 [DEBUG.SRC]DBGNPNP.B32;1

```

Page 55  
(18)

04	51		52	C0	001B6		ADDL2	R2, R1
D4	60		51	A2	001B9		SUBW2	R1, (R0)
	A0	04	B842	9E	001BC		MOVAB	@LEX_STRING_DESC+4[R2], 4(R0)
	A8	F8	A8	D0	001C2		MOVL	TOKEN, LAST_TOKEN
	50		01	D0	001C7	18\$:	MOVL	#1, R0
				04	001CA		RET	

: 1667 1783 1



```
: 1669      1784 1 ROUTINE ID_ITEM =
: 1670      1785 1
: 1671      1786 1 ++
: 1672      1787 1 FUNCTIONAL DESCRIPTION:
: 1673      1788 1
: 1674      1789 1     Parses an ID item.
: 1675      1790 1
: 1676      1791 1 FORMAL PARAMETERS:
: 1677      1792 1
: 1678      1793 1     NONE
: 1679      1794 1
: 1680      1795 1 IMPLICIT INPUTS:
: 1681      1796 1
: 1682      1797 1     The augmentation vector.
: 1683      1798 1
: 1684      1799 1 IMPLICIT OUTPUTS:
: 1685      1800 1
: 1686      1801 1     NONE
: 1687      1802 1
: 1688      1803 1 ROUTINE VALUE:
: 1689      1804 1
: 1690      1805 1     An unsigned integer longword completion code
: 1691      1806 1
: 1692      1807 1 COMPLETION CODES:
: 1693      1808 1
: 1694      1809 1     STS$K_SUCCESS           - Success. Valid ID item parsed.
: 1695      1810 1
: 1696      1811 1     STS$K_SEVERE           - Failure. Invalid ID item found.
: 1697      1812 1
: 1698      1813 1 SIDE EFFECTS:
: 1699      1814 1
: 1700      1815 1     The ID item is added to the pathname descriptor.
: 1701      1816 1
: 1702      1817 1 --
: 1703      1818 2 BEGIN
: 1704      1819 2
: 1705      1820 2     add_id;
: 1706      1821 2     advance;
: 1707      1822 2     get_token;
: 1708      1823 2
: 1709      1824 2
: 1710      1825 2     ! Check for invocation number
: 1711      1826 2     !
: 1712      1827 2     IF .token EQL dbg$tok_tok_int
: 1713      1828 2     THEN
: 1714      1829 2         BEGIN ! See if an invocation number has already been found.
: 1715      1830 2             IF .augmentations [invocation_found] THEN RETURN sts$severe;
: 1716      1831 2             add_invocation_number;
: 1717      1832 2             advance;
: 1718      1833 2             END;
: 1719      1834 2
: 1720      1835 2     RETURN sts$success;
: 1721      1836 2
: 1722      1837 1     END;           ! End of ID_ITEM
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

OD 00004 P.AAE: .BYTE 13

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

				01FC 00000	ID_ITEM:	.WORD	Save R2,R3,R4,R5,R6,R7,R8	1784	
		58	00000000G	00	9E	00002	MOVAB	DBG\$GET_TEMPMEM, R8	
		57	000000000	EF	9E	00009	MOVAB	LEX_STRING_DESC, R7	
		5E		10	C2	00010	SUBL2	#16, SP	
		50		67	3C	00013	MOVZWL	LEX_STRING_DESC, R0	1818
		50		04	C6	00016	DIVL2	#4, R0	
			01	A0	9F	00019	PUSHAB	1(R0)	
		68		01	FB	0001C	CALLS	#1, DBG\$GET_TEMPMEM	
01	A6	56		50	D0	0001F	MOVL	R0, NAME_STRING	
		B7		67	28	00022	MOV3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -	
								1(NAME_STRING)	
		66		67	90	00028	MOVB	LEX_STRING_DESC, (NAME_STRING)	
		52		A7	D0	0002B	MOVL	NAME_INDEX, R2	
		32		52	D1	0002F	CMPL	R2, #50	
				0F	19	00032	BLSS	1\$	
			00028200	8F	DD	00034	PUSHL	#164352	
			00	01	FB	0003A	CALLS	#1, LIB\$SIGNAL	
				08	11	00041	BRB	2\$	
		E4	B742	56	D0	00043	MOVL	NAME_STRING, @NAME_VECTOR[R2]	
				A7	D6	00048	INCL	NAME_INDEX	
		50		A7	D0	0004B	MOVL	PATHNAME_DESC, R0	
				60	96	0004F	INCB	(R0)	
		01	A0	60	90	00051	MOVB	(R0), 1(R0)	
C8	A7	67		08	28	00055	MOV3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1820
		51		A7	D0	0005A	MOVL	INPUT_DESC, R1	
	50	04	A7	A1	C3	0005E	SUBL3	4(R1), LEX_STRING_DESC+4, R0	
		52		67	3C	00064	MOVZWL	LEX_STRING_DESC, R2	
		50		52	C0	00067	ADDL2	R2, R0	
		61		50	A2	0006A	SUBW2	R0, (R1)	
		04	A1	04	B742	9E	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	
		D4	A7	F8	A7	D0	MOVL	TOKEN, LAST_TOKEN	
				F8	A7	9F	PUSHAB	TOKEN	1821
				8F	BB	0007B	PUSHR	#*M<R1,R7>	
		FC	B7	03	FB	0007F	CALLS	#3, @TOKEN_SCANNER_ADDR	
		06		A7	D1	00083	CMPL	TOKEN, #6	
			09	09	12	00087	BNEQ	3\$	
			09	67	B1	00089	CMPL	LEX_STRING_DESC, #9	
				04	1B	0008C	BLEQU	3\$	
		F8	A7	01	D0	0008E	MOVL	#1, TOKEN	
		06		A7	D1	00092	CMPL	TOKEN, #6	1827
				75	12	00096	BNEQ	6\$	
	3C	F4	A7	04	E0	00098	BBS	#4, AUGMENTATIONS, 4\$	1830
		F4	A7	10	88	0009D	BISB2	#16, AUGMENTATIONS	
04	AE	67		01	A1	000A1	ADDW3	#1, LEX_STRING_DESC, NUMBER_DESC	
		50		AE	3C	000A6	MOVZWL	NUMBER_DESC, R0	
		50		04	C6	000AA	DIVL2	#4, R0	
			01	A0	9F	000AD	PUSHAB	1(R0)	
		68		01	FB	000B0	CALLS	#1, DBG\$GET_TEMPMEM	
		56		50	D0	000B3	MOVL	R0, NUM_BUF	



66	04	B7		67	28	000B6	MOVC3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -	:
		63	00000000'	EF	90	000BB	MOVB	(NUM_BUF)	:
	08	AE		56	D0	000C2	MOVL	P.AAE, (POINTER)	:
			D8	A7	9F	000C6	PUSHAB	NUM_BUF, NUMBER_DESC+4	:
			04	AE	9F	000C9	PUSHAB	DUMMY	:
			0C	AE	9F	000CC	PUSHAB	NUMBER	:
				03	FB	000CF	PUSHAB	NUMBER_DESC	:
	00000000G	00		50	E8	000D6	CALLS	#3, DBG\$NSAVE_DECIMAL_INTEGER	:
		04		04	D0	000D9	BLBS	R0, 5\$	:
		50		04	D0	000D9	MOVL	#4, R0	:
				04	D0	000DC	RET		:
		50	E0	A7	D0	000DD	MOVL	PATHNAME_DESC, R0	:
	02	A0	E8	A7	90	000E1	MOVB	NAME_INDEX, 2(R0)	:
	04	A0		6E	D0	000E6	MOVL	NUMBER, 4(R0)	:
C8	A7	67		08	28	000EA	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1831
		50	DC	A7	D0	000EF	MOVL	INPUT_DESC, R0	:
51	04	A7	04	A0	C3	000F3	SUBL3	4(R0), LEX_STRING_DESC+4, R1	:
		52		67	3C	000F9	MOVZWL	LEX_STRING_DESC, R2	:
		51		52	C0	000FC	ADDL2	R2, R1	:
		60		51	A2	000FF	SUBW2	R1, (R0)	:
	04	A0	04	B742	9E	00102	MOVAB	@LEX_STRING_DESC+4[R2], 4(R0)	:
	D4	A7	F8	A7	D0	00108	MOVL	TOKEN, LAST_TOKEN	:
		50		01	D0	0010D	MOVL	#1, R0	1835
				04	00	00110	RET		1837

; Routine Size: 273 bytes, Routine Base: DBG\$CODE + 0A8D

; 1723 1838 1

```
1725 1839 1 ROUTINE INTEGER_ITEM =
1726 1840 1
1727 1841 1 ++
1728 1842 1 FUNCTIONAL DESCRIPTION:
1729 1843 1
1730 1844 1     Parses a dangling line or label number.
1731 1845 1
1732 1846 1 FORMAL PARAMETERS:
1733 1847 1
1734 1848 1     NONE
1735 1849 1
1736 1850 1 IMPLICIT INPUTS:
1737 1851 1
1738 1852 1     The augmentation vector.
1739 1853 1
1740 1854 1 IMPLICIT OUTPUTS:
1741 1855 1
1742 1856 1     NONE
1743 1857 1
1744 1858 1 ROUTINE VALUE:
1745 1859 1
1746 1860 1     An unsigned integer longword completion code
1747 1861 1
1748 1862 1 COMPLETION CODES:
1749 1863 1
1750 1864 1     STSSK_SUCCESS           - Success. LINE or LABEL number parsed.
1751 1865 1
1752 1866 1     STSSK_SEVERE           - Failure. Invalid integer item found.
1753 1867 1
1754 1868 1 SIDE EFFECTS:
1755 1869 1
1756 1870 1     The line or label number is added to the pathname descriptor.
1757 1871 1
1758 1872 1 --
1759 1873 2 BEGIN
1760 1874 2
1761 1875 2     ! Determine if looking for line or label number
1762 1876 2
1763 1877 2 SELECTONE true
1764 1878 2     OF
1765 1879 2     SET
1766 1880 2
1767 1881 2     [.augmentations [line_pending]] :      ! Line number
1768 1882 2     BEGIN
1769 1883 2     add_to_l_number;
1770 1884 2     advance;
1771 1885 2     get_token;
1772 1886 2
1773 1887 2
1774 1888 2     ! See if more line number follows
1775 1889 2
1776 1890 2     IF .token EQL dbg$tok_tok_dot
1777 1891 2     THEN
1778 1892 2     BEGIN
1779 1893 2     add_to_l_number;
1780 1894 2     advance;
1781 1895 2     get_token;
```



```
: 1782      1896  4
: 1783      1897  4
: 1784      1898  4
: 1785      1899  4
: 1786      1900  4
: 1787      1901  4
: 1788      1902  4
: 1789      1903  4
: 1790      1904  4
: 1791      1905  4
: 1792      1906  4
: 1793      1907  4
: 1794      1908  4
: 1795      1909  4
: 1796      1910  4
: 1797      1911  4
: 1798      1912  4
: 1799      1913  4
: 1800      1914  4
: 1801      1915  4
: 1802      1916  4
: 1803      1917  4
: 1804      1918  4
: 1805      1919  4
: 1806      1920  4
: 1807      1921  4
: 1808      1922  4
: 1809      1923  1

      IF .token NEQ dbg$tok_int THEN RETURN sts$k_severe;
      add_to_l_number;
      add_line;
      advance;
      END
    ELSE
      add_line;
    END;

[.augmentations [label_pending]] :      ! LABEL number
  BEGIN
    add_to_l_number;
    add_label;
    advance;
  END;

[OTHERWISE] :
  RETURN sts$k_severe;

TES;

augmentations [terminal_pending] = true;

RETURN sts$k_success;

END;      ! End of INTEGER_ITEM
```

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

```
20 20 45 4E 49 4C 25 00005 P.AAF: .ASCII \%LINE \
20 20 45 4E 49 4C 25 0000B P.AAG: .ASCII \%LINE \
20 4C 45 42 41 4C 25 00011 P.AAH: .ASCII \%LABEL \
```

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

```
OFFC 00000 INTEGER_ITEM:
5B 00000000G 00 9E 00002 .WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11 : 1839
5A 00000000' EF 9E 00009 MOVAB DBG$GET_TEMPMEM, R11
5E          OC C2 00010 MOVAB NUMBER_BUFFER, R10
03          04 AA E8 00013 SUBL2 #12, SP
027F        31 00017 BLBS AUGMENTATIONS, 1$ : 1881
04 AE        14 AA D0 0001A 1$: BRW 21$
6E          10 AA B0 0001F MOVW LEX_STRING_DESC+4, NUMBER_DESC+4 : 1882
01          6E B1 00023 2$: CMPW LEX_STRING_DESC, NUMBER_DESC
OD          1B 00026 BLEQU NUMBER_DESC, #1
30          04 BE 91 00028 CMPB @NUMBER_DESC+4, #48
07          12 0002C BNEQ 3$
6E          B7 0002E DECW NUMBER_DESC
04          AE D6 00030 INCL NUMBER_DESC+4
EE          11 00033 BRB 2$
```

31	04	58	6E	3C	00035	3\$:	MOVZWL	NUMBER_DESC, R8		
		AA	05	E1	00038		BBC	#5, AUGMENTATIONS, 4\$		
		56	6A	D0	0003D		MOVL	NUMBER_BUFFER, TEMP		
		59	66	9A	00040		MOVZBL	(TEMP), R9		
		59	58	C0	00043		ADDL2	R8, R9		
50		59	04	C7	00046		DIVL3	#4, R9, R0		
			01	A0	9F	0004A	PUSHAB	1(R0)		
		6B	01	FB	0004D		CALLS	#1, DBG\$GET_TEMPMEM		
		6A	50	D0	00050		MOVL	R0, NUMBER_BUFFER		
		50	66	9A	00053		MOVZBL	(TEMP), R0		
		57	6A	D0	00056		MOVL	NUMBER_BUFFER, R7		
01 A7	01	A6	50	28	00059		MOVC3	R0, 1(TEMP), 1(R7)		
		50	66	9A	0005F		MOVZBL	(TEMP), R0		
01 A047	04	BE	58	28	00062		MOVC3	R8, @NUMBER_DESC+4, 1(R0)[R7]		
		67	59	90	00069		MOVB	R9, (R7)		
			1D	11	0006C		BRB	5\$		
		AA	20	88	0006E	4\$:	BISB2	#32, AUGMENTATIONS		
50	04	58	04	C7	00072		DIVL3	#4, R8, R0		
			01	A0	9F	00076	PUSHAB	1(R0)		
		6B	01	FB	00079		CALLS	#1, DBG\$GET_TEMPMEM		
		6A	50	D0	0007C		MOVL	R0, NUMBER_BUFFER		
		56	6A	D0	0007F		MOVL	NUMBER_BUFFER, R6		
01 A6	04	BE	58	28	00082		MOVC3	R8, @NUMBER_DESC+4, 1(R6)		
		66	58	90	00088		MOVB	R8, (R6)		
D8 AA	10	AA	08	28	0008B	5\$:	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1883	
		51	EC	AA	D0	00091	MOVL	INPUT_DESC, R1		
		52	04	A1	C3	00095	SUBL3	4(R1), LEX_STRING_DESC+4, R0		
50	14	AA	10	AA	3C	0009B	MOVZWL	LEX_STRING_DESC, R2		
		50		52	C0	0009F	ADDL2	R2, R0		
		61		50	A2	000A2	SUBW2	R0, (R1)		
	04	A1	14	BA	42	9E	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)		
	E4	AA	08	AA	D0	000AB	MOVL	TOKEN, LAST_TOKEN		
			08	AA	9F	000B0	PUSHAB	TOKEN	1884	
			10	AA	9F	000B3	PUSHAB	LEX_STRING_DESC		
				51	DD	000B6	PUSHL	R1		
	0C	BA	03	FB	000B8		CALLS	#3, @TOKEN_SCANNER_ADDR		
		06	08	AA	D1	000BC	CMPL	TOKEN, #6		
				0A	12	000C0	BNEQ	6\$		
		09	10	AA	B1	000C2	CMPL	LEX_STRING_DESC, #9		
				04	1B	000C6	BLEQU	6\$		
	08	AA	01	D0	000C8		MOVL	#1, TOKEN		
		07	08	AA	D1	0C0CC	CMPL	TOKEN, #7	1890	
				03	13	000D0	BEQL	7\$		
			01	63	31	000D2	BRW	18\$		
	04	AE	14	AA	D0	000D5	7\$:	MOVL	LEX_STRING_DESC+4, NUMBER_DESC+4	1892
		6E	10	AA	B0	000DA	MOVW	LEX_STRING_DESC, NUMBER_DESC		
		01		6E	B1	000DE	8\$:	CMPL	NUMBER_DESC, #1	
				0D	1B	000E1	BLEQU	9\$		
		30	04	BE	91	000E3	CMPB	@NUMBER_DESC+4, #48		
				07	12	000E7	BNEQ	9\$		
				6E	B7	000E9	DECW	NUMBER_DESC		
			04	AE	D6	000EB	INCL	NUMBER_DESC+4		
				EE	11	000EE	BRB	8\$		
		58		6E	3C	000F0	9\$:	MOVZWL	NUMBER_DESC, R8	
31	04	AA	05	E1	000F3		BBC	#5, AUGMENTATIONS, 10\$		
		56	6A	D0	000F8		MOVL	NUMBER_BUFFER, TEMP		
		59	66	9A	000FB		MOVZBL	(TEMP), R9		



50	59	58	C0	000FE	ADDL2	R8, R9	
	59	04	C7	00101	DIVL3	#4, R9, R0	
	6B	01	A0	9F 00105	PUSHAB	1(R0)	
	6A	01	FB	00108	CALLS	#1, DBG\$GET_TEMP MEM	
	50	50	D0	0010B	MOVL	R0, NUMBER_BUFFER	
	50	66	9A	0010E	MOVZBL	(TEMP), R0	
01 A7	57	6A	D0	00111	MOVL	NUMBER_BUFFER, R7	
	A6	50	28	00114	MOVC3	R0, 1(TEMP), 1(R7)	
01 A047	50	66	9A	0011A	MOVZBL	(TEMP), R0	
	BE	58	28	0011D	MOVC3	R8, @NUMBER_DESC+4, 1(R0)[R7]	
	67	59	90	00124	MOVB	R9, (R7)	
		1D	11	00127	BRB	11\$	
50	04 AA	20	88	00129	BISB2	#32, AUGMENTATIONS	
	58	04	C7	0012D	DIVL3	#4, R8, R0	
	6B	01	A0	9F 00131	PUSHAB	1(R0)	
	6A	01	FB	00134	CALLS	#1, DBG\$GET_TEMP MEM	
	56	50	D0	00137	MOVL	R0, NUMBER_BUFFER	
01 A6	04 BE	6A	D0	0013A	MOVL	NUMBER_BUFFER, R6	
	66	58	28	0013D	MOVC3	R8, @NUMBER_DESC+4, 1(R6)	
D8 AA	10 AA	58	90	00143	MOVB	R8, (R6)	
	51	08	28	00146	MOVC3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	1893
50	14 AA	EC	AA	D0 0014C	MOVL	INPUT_DESC, R1	
	52	04	A1	C3 00150	SUBL3	4(R1), LEX_STRING_DESC+4, R0	
	50	10	AA	3C 00156	MOVZWL	LEX_STRING_DESC, R2	
	61		52	C0 0015A	ADDL2	R2, R0	
	04 A1	14	50	A2 0015D	SUBW2	R0, (R1)	
E4 AA		08	BA42	9E 00160	MOVAB	@LEX_STRING_DESC+4[R2], 4(R1)	
		08	AA	D0 00166	MOVL	TOKEN, LAST_TOKEN	
		08	AA	9F 0016B	PUSHAB	TOKEN	1894
		10	AA	9F 0016E	PUSHAB	LEX_STRING_DESC	
			51	DD 00171	PUSHL	R1	
	0C BA	03	FB	00173	CALLS	#3, @TOKEN_SCANNER_ADDR	
	06	08	AA	D1 00177	CMPL	TOKEN, #6	
			0A	12 0017B	BNEQ	12\$	
	09	10	AA	B1 0017D	CMPL	LEX_STRING_DESC, #9	
			04	1B 00181	BLEQU	12\$	
	08 AA	01	D0	00183	MOVL	#1, TOKEN	
	06	08	AA	D1 00187	CMPL	TOKEN, #6	1897
			03	13 0018B	BEQL	13\$	
		0207	31	0018D	BRW	30\$	
	04 AE	14	AA	D0 00190	MOVL	LEX_STRING_DESC+4, NUMBER_DESC+4	
	6E	10	AA	B0 00195	MOVW	LEX_STRING_DESC, NUMBER_DESC	
	01		6E	B1 00199	CMPL	NUMBER_DESC, #1	
			0D	1B 0019C	BLEQU	15\$	
	30	04	BE	91 0019E	CMPL	@NUMBER_DESC+4, #48	
			07	12 001A2	BNEQ	15\$	
			6E	B7 001A4	DECW	NUMBER_DESC	
		04	AE	D6 001A6	INCL	NUMBER_DESC+4	
			EE	11 001A9	BRB	14\$	
31	04 58		6E	3C 001AB	MOVZWL	NUMBER_DESC, R8	
	AA		05	E1 001AE	BBC	#5, AUGMENTATIONS, 16\$	
	56		6A	D0 001B3	MOVL	NUMBER_BUFFER, TEMP	
	59		66	9A 001B6	MOVZBL	(TEMP), R9	
	59		58	C0 001B9	ADDL2	R8, R9	
50	59		04	C7 001BC	DIVL3	#4, R9, R0	
		01	A0	9F 001C0	PUSHAB	1(R0)	
	6B		01	FB 001C3	CALLS	#1, DBG\$GET_TEMP MEM	

01	A7	01	6A	50	D0	001C6	MOVL	R0, NUMBER_BUFFER	
			50	66	9A	001C9	MOVZBL	(TEMP), R0	
			57	6A	D0	001CC	MOVL	NUMBER_BUFFER, R7	
01	A047	04	A6	50	28	001CF	MOVC3	R0, 1(TEMP), 1(R7)	
			50	66	9A	001D5	MOVZBL	(TEMP), R0	
			BE	58	28	001D8	MOVC3	R8, @NUMBER_DESC+4, 1(R0)[R7]	
			67	59	90	001DF	MOVB	R9, (R7)	
				1D	11	001E2	BRB	17\$	
	50	04	AA	20	88	001E4	BISB2	#32, AUGMENTATIONS	
			58	04	C7	001E8	DIVL3	#4, R8, R0	
				01	A0	9F	001EC	PUSHAB	1(R0)
			6B	01	FB	001EF	CALLS	#1, DBG\$GET TEMPMEM	
			6A	50	D0	001F2	MOVL	R0, NUMBER_BUFFER	
01	A6	04	56	6A	D0	001F5	MOVL	NUMBER_BUFFER, R6	
			BE	58	28	001F8	MOVC3	R8, @NUMBER_DESC+4, 1(R6)	
			66	58	90	001FE	MOVB	R8, (R6)	
		04	AA	02	88	00201	BISB2	#2, AUGMENTATIONS	1899
		04	AA	01	8A	00205	BICB2	#1, AUGMENTATIONS	
			50	00	BA	9A	00209	MOVZBL	@NUMBER_BUFFER, R0
			50	06	C0	0020D	ADDL2	#6, R0	
			50	04	C6	00210	DIVL2	#4, R0	
				01	A0	9F	00213	PUSHAB	1(R0)
			6B	01	FB	00216	CALLS	#1, DBG\$GET TEMPMEM	
01	A7 00000000'		57	50	D0	00219	MOVL	R0, LINE_ITEM	
			EF	06	28	0021C	MOVC3	#6, P.AAF, 1(LINE_ITEM)	
			56	6A	D0	00225	MOVL	NUMBER_BUFFER, R6	
07	A7	01	50	66	9A	00228	MOVZBL	(R6), R0	
	67		A6	50	28	0022B	MOVC3	R0, 1(R6), 7(LINE_ITEM)	
			66	06	81	00231	ADDB3	#6, (R6), (LINE_ITEM)	
				01	0E	31	00235	BRW	27\$
			AA	02	88	00238	BISB2	#2, AUGMENTATIONS	1903
		04	AA	01	8A	0023C	BICB2	#1, AUGMENTATIONS	
			50	00	BA	9A	00240	MOVZBL	@NUMBER_BUFFER, R0
			50	06	C0	00244	ADDL2	#6, R0	
			50	04	C6	00247	DIVL2	#4, R0	
				01	A0	9F	0024A	PUSHAB	1(R0)
			6B	01	FB	0024D	CALLS	#1, DBG\$GET TEMPMEM	
01	A7 00000000'		57	50	D0	00250	MOVL	R0, LINE_ITEM	
			EF	06	28	00253	MOVC3	#6, P.AAG, 1(LINE_ITEM)	
			56	6A	D0	0025C	MOVL	NUMBER_BUFFER, R6	
07	A7	01	50	66	9A	0025F	MOVZBL	(R6), R0	
	67		A6	50	28	00262	MOVC3	R0, 1(R6), 7(LINE_ITEM)	
			66	06	81	00268	ADDB3	#6, (R6), (LINE_ITEM)	
			52	F8	AA	D0	0026C	MOVL	NAME_INDEX, R2
			32	52	D1	00270	CMPL	R2, #50	
				0F	19	00273	BLSS	19\$	
				8F	DD	00275	PUSHL	#164352	
	00000000G	00	01	01	FB	0027B	CALLS	#1, LIB\$SIGNAL	
			08	11	00282	BRB	20\$		
			57	D0	00284	MOVL	LINE_ITEM, @NAME_VECTOR[R2]	19\$:	
		F4 BA42		AA	D6	00289	INCL	NAME_INDEX	
			50	AA	D0	0028C	MOVL	PATHNAME_DESC, R0	20\$:
				60	96	00290	INCB	(R0)	
		01	A0	60	90	00292	MOVB	(R0), 1(R0)	
				01	02	31	00296	BRW	31\$
03		04	AA	02	E0	00299	BBS	#2, AUGMENTATIONS, 22\$	1877
				00F6	31	0029E	BRW	30\$	1907



	04	AE	14	AA	D0	002A1	22\$:	MOVL	LEX_STRING_DESC+4, NUMBER_DESC+4	
	6E	10	AA	B0	002A6			MOVW	LEX_STRING_DESC, NUMBER_DESC	
	01		6E	B1	002AA	23\$:		CMPW	NUMBER_DESC, #1	
	30		0D	1B	002AD			BLEQU	24\$	
			04	BE	91	002AF		CMPB	@NUMBER_DESC+4, #48	
			07	12	002B3			BNEQ	24\$	
			6E	B7	002B5			DECW	NUMBER_DESC	
			04	AE	D6	002B7		INCL	NUMBER_DESC+4	
			EE	11	002BA			BRB	23\$	
		58	6E	3C	002BC	24\$:		MOVZWL	NUMBER_DESC, R8	
31	04	AA	05	E1	002BF			BBC	#5, AUGMENTATIONS, 25\$	
		56	6A	D0	002C4			MOVL	NUMBER_BUFFER, TEMP	
		59	66	9A	002C7			MOVZBL	(TEMP), R9	
		59	58	C0	002CA			ADDL2	R8, R9	
50		59	04	C7	002CD			DIVL3	#4, R9, R0	
			01	A0	9F	002D1		PUSHAB	1(R0)	
		6B	01	FB	002D4			CALLS	#1, DBG\$GET TEMPMEM	
		6A	50	D0	002D7			MOVL	R0, NUMBER_BUFFER	
		50	66	9A	002DA			MOVZBL	(TEMP), R0	
		57	6A	D0	002DD			MOVL	NUMBER_BUFFER, R7	
01 A7	01	A6	50	28	002E0			MOVC3	R0, 1(TEMP), 1(R7)	
		50	66	9A	002E6			MOVZBL	(TEMP), R0	
01 A047	04	BE	58	28	002E9			MOVC3	R8, @NUMBER_DESC+4, 1(R0)[R7]	
		67	59	90	002F0			MOVB	R9, (R7)	
			1D	11	002F3			BRB	26\$	
		50	04	AA	20	88	002F5	25\$:	BISB2	#32, AUGMENTATIONS
		58	04	C7	002F9			DIVL3	#4, R8, R0	
			01	A0	9F	002FD		PUSHAB	1(R0)	
		6B	01	FB	00300			CALLS	#1, DBG\$GET TEMPMEM	
		6A	50	D0	00303			MOVL	R0, NUMBER_BUFFER	
		57	6A	D0	00306			MOVL	NUMBER_BUFFER, R7	
01 A7	04	BE	58	28	00309			MOVC3	R8, @NUMBER_DESC+4, 1(R7)	
		67	58	90	0030F			MOVB	R8, (R7)	
		04	08	88	00312	26\$:		BISB2	#8, AUGMENTATIONS	
		04	04	8A	00316			BICB2	#4, AUGMENTATIONS	
		50	00	BA	9A	0031A		MOVZBL	@NUMBER_BUFFER, R0	
		50	07	C0	0031E			ADDL2	#7, R0	
		50	04	C6	00321			DIVL2	#4, R0	
			01	A0	9F	00324		PUSHAB	1(R0)	
		6B	01	FB	00327			CALLS	#1, DBG\$GET TEMPMEM	
01 A7 00000000'		57	50	D0	0032A			MOVL	R0, LABEL_ITEM	
		EF	07	28	0032D			MOVC3	#7, P.AAH, 1(LABEL_ITEM)	
		56	6A	D0	00336			MOVL	NUMBER_BUFFER, R6	
		50	66	9A	00339			MOVZBL	(R6), R0	
08 A7	01	A6	50	28	0033C			MOVC3	R0, 1(R6), 8(LABEL_ITEM)	
67		66	07	81	00342			ADDB3	#7, (R6), (LABEL_ITEM)	
		52	F8	AA	D0	00346	27\$:	MOVL	NAME_INDEX, R2	
		32	52	D1	0034A			CMPL	R2, #50	
			0F	19	0034D			BLSS	28\$	
			8F	DD	0034F			PUSHL	#164352	
00000000G	00		01	FB	00355			CALLS	#1, LIB\$SIGNAL	
			08	11	0035C			BRB	29\$	
		F4 BA42	57	D0	0035E	28\$:		MOVL	LABEL_ITEM, @NAME_VECTOR[2]	
			F8	AA	D6	00363		INCL	NAME_INDEX	
		50	F0	AA	D0	00366	29\$:	MOVL	PATHNAME_DESC, R0	
			60	96	0036A			INCB	(R0)	
		01 A0	60	90	0036C			MOVB	(R0), 1(R0)	

1908

1909

DBGNPNP  
V04-000

N 5  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGNPNP.B32;1

Page 65  
(20)

D8	AA	10	AA	08	28	00370	MOV C3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	: 1910
			50	EC	AA	D0 00376	MOVL	INPUT_DESC, R0	:
	51	14	AA	04	A0	C3 0037A	SUB L3	4(R0), LEX_STRING_DESC+4, R1	:
			52	10	AA	3C 00380	MOVZWL	LEX_STRING_DESC, R2	:
			51		52	C0 00384	ADD L2	R2, R1	:
			60		51	A2 00387	SUB W2	R1, (R0)	:
		04	A0	14	BA42	9E 0038A	MOVAB	@LEX_STRING_DESC+4[R2], 4(R0)	:
		E4	AA	08	AA	D0 00390	MOVL	TOKEN, LAST_TOKEN	:
			50		04	11 00395	BRB	31\$	: 1877
					04	D0 00397	MOVL	#4, R0	: 1915
					04	0039A	RET		:
		04	AA	40	8F	88 0039B	BISB2	#64, AUGMENTATIONS	: 1919
			50		01	D0 003A0	MOVL	#1, R0	: 1921
					04	003A3	RET		: 1923

; Routine Size: 932 bytes, Routine Base: DBG\$CODE + 0B9E

; 1810 1924 1



```
1812 1925 1 ROUTINE SHORT_SCOPE =
1813 1926 1
1814 1927 1 ++
1815 1928 1 FUNCTIONAL DESCRIPTION:
1816 1929 1
1817 1930 1     Parses global or numeric scopes. On failure, resets input to original state.
1818 1931 1
1819 1932 1 FORMAL PARAMETERS:
1820 1933 1
1821 1934 1     NONE
1822 1935 1
1823 1936 1 IMPLICIT INPUTS:
1824 1937 1
1825 1938 1     NONE
1826 1939 1
1827 1940 1 IMPLICIT OUTPUTS:
1828 1941 1
1829 1942 1     NONE
1830 1943 1
1831 1944 1 ROUTINE VALUE:
1832 1945 1
1833 1946 1     An unsigned integer longword completion code
1834 1947 1
1835 1948 1 COMPLETION CODES:
1836 1949 1
1837 1950 1     STS$K_SUCCESS           - Success. Global or numeric scope accepted.
1838 1951 1
1839 1952 1     STS$K_SEVERE           - Failure. Input not a numeric or global scope
1840 1953 1
1841 1954 1 SIDE EFFECTS:
1842 1955 1
1843 1956 1     If successful, produces a complete pathname descriptor for global
1844 1957 1     or numeric scope.
1845 1958 1
1846 1959 1 --
1847 1960 2 BEGIN
1848 1961 2
1849 1962 2 LOCAL
1850 1963 2     LENGTH,           ! Original input length
1851 1964 2     POINTER;       ! Original input pointer
1852 1965 2
1853 1966 2     ! Save the original input
1854 1967 2     !
1855 1968 2     save (length, pointer);
1856 1969 2
1857 1970 2
1858 1971 2     ! Obtain the first token and check for integer or backslash
1859 1972 2     !
1860 1973 2     get_token;
1861 1974 2
1862 1975 2 CASE .token FROM dbg$k_tok_lowest TO dbg$k_tok_highest
1863 1976 2     OF
1864 1977 2     SET
1865 1978 2
1866 1979 2     [dbg$k_tok_bs] : ! Global scope ?
1867 1980 2     BEGIN
1868 1981 2     advance;
```



```

: 1869      1982  3      get_token;
: 1870      1983  3
: 1871      1984  3      IF .token EQL dbg$k_tok_null OR .token EQL dbg$k_tok_inval
: 1872      1985  3      THEN
: 1873      1986  4          BEGIN
: 1874      1987  4              ! Yes, global scope.
: 1875      1988  4              !
: 1876      1989  4              add_null_id;
: 1877      1990  4              END
: 1878      1991  4          ELSE
: 1879      1992  3              BEGIN
: 1880      1993  4                  ! No. Restore input.
: 1881      1994  4                  !
: 1882      1995  4                  restore (.length, .pointer);
: 1883      1996  4                  RETURN sts$k_severe;
: 1884      1997  4                  END;
: 1885      1998  4              END;
: 1886      1999  3          END;
: 1887      2000  2
: 1888      2001  2      [dbg$k_tok_int] :      ! Numeric scope ?
: 1889      2002  2      BEGIN
: 1890      2003  3          advance;
: 1891      2004  3          get_token;
: 1892      2005  3
: 1893      2006  3      IF .token EQL dbg$k_tok_inval OR .token EQL dbg$k_tok_null
: 1894      2007  3      THEN
: 1895      2008  3          BEGIN
: 1896      2009  4              ! Yes, numeric scope
: 1897      2010  4              !
: 1898      2011  4              restore (.length, .pointer);
: 1899      2012  4              get_token;
: 1900      2013  4              add_numeric_scope;
: 1901      2014  4              advance;
: 1902      2015  4              END
: 1903      2016  4          ELSE
: 1904      2017  4              BEGIN
: 1905      2018  3                  ! No, restore and fail
: 1906      2019  4                  !
: 1907      2020  4                  restore (.length, .pointer);
: 1908      2021  4                  RETURN sts$k_severe;
: 1909      2022  4                  END;
: 1910      2023  4              END;
: 1911      2024  4          END;
: 1912      2025  3      END;
: 1913      2026  2
: 1914      2027  2      [INRANGE, OUTRANGE] :
: 1915      2028  2      BEGIN
: 1916      2029  3          RETURN sts$k_severe;
: 1917      2030  3          END;
: 1918      2031  2
: 1919      2032  2      TES;
: 1920      2033  2      RETURN sts$k_success;
: 1921      2034  2
: 1922      2035  2
: 1923      2036  2
: 1924      2037  1      END;      ! End of short_scope
```



```
.PSECT DBG$CODE,NOWRT, SHR, PIC,0
```

```

WORD Save R2,R3,R4,R5,R6,R7,R8,R9,R10
MOVAB NULL_STRING, R10

```

[illegible]

PC	Op	Op2	Op3	Op4	Op5	Op6	Op7	Op8	Op9	Op10	Op11	Op12	Op13	Op14	Op15	Op16	Op17	Op18	Op19	Op20	Op21	Op22	Op23	Op24	Op25	Op26	Op27	Op28	Op29	Op30	Op31	Op32	Op33	Op34	Op35	Op36	Op37	Op38	Op39	Op40	Op41	Op42	Op43	Op44	Op45	Op46	Op47	Op48	Op49	Op50	Op51	Op52	Op53	Op54	Op55	Op56	Op57	Op58	Op59	Op60	Op61	Op62	Op63	Op64	Op65	Op66	Op67	Op68	Op69	Op70	Op71	Op72	Op73	Op74	Op75	Op76	Op77	Op78	Op79	Op80	Op81	Op82	Op83	Op84	Op85	Op86	Op87	Op88	Op89	Op90	Op91	Op92	Op93	Op94	Op95	Op96	Op97	Op98	Op99	Op100	Op101	Op102	Op103	Op104	Op105	Op106	Op107	Op108	Op109	Op110	Op111	Op112	Op113	Op114	Op115	Op116	Op117	Op118	Op119	Op120	Op121	Op122	Op123	Op124	Op125	Op126	Op127	Op128	Op129	Op130	Op131	Op132	Op133	Op134	Op135	Op136	Op137	Op138	Op139	Op140	Op141	Op142	Op143	Op144	Op145	Op146	Op147	Op148	Op149	Op150	Op151	Op152	Op153	Op154	Op155	Op156	Op157	Op158	Op159	Op160	Op161	Op162	Op163	Op164	Op165	Op166	Op167	Op168	Op169	Op170	Op171	Op172	Op173	Op174	Op175	Op176	Op177	Op178	Op179	Op180	Op181	Op182	Op183	Op184	Op185	Op186	Op187	Op188	Op189	Op190	Op191	Op192	Op193	Op194	Op195	Op196	Op197	Op198	Op199	Op200	Op201	Op202	Op203	Op204	Op205	Op206	Op207	Op208	Op209	Op210	Op211	Op212	Op213	Op214	Op215	Op216	Op217	Op218	Op219	Op220	Op221	Op222	Op223	Op224	Op225	Op226	Op227	Op228	Op229	Op230	Op231	Op232	Op233	Op234	Op235	Op236	Op237	Op238	Op239	Op240	Op241	Op242	Op243	Op244	Op245	Op246	Op247	Op248	Op249	Op250	Op251	Op252	Op253	Op254	Op255	Op256	Op257	Op258	Op259	Op260	Op261	Op262	Op263	Op264	Op265	Op266	Op267	Op268	Op269	Op270	Op271	Op272	Op273	Op274	Op275	Op276	Op277	Op278	Op279	Op280	Op281	Op282	Op283	Op284	Op285	Op286	Op287	Op288	Op289	Op290	Op291	Op292	Op293	Op294	Op295	Op296	Op297	Op298	Op299	Op300	Op301	Op302	Op303	Op304	Op305	Op306	Op307	Op308	Op309	Op310	Op311	Op312	Op313	Op314	Op315	Op316	Op317	Op318	Op319	Op320	Op321	Op322	Op323	Op324	Op325	Op326	Op327	Op328	Op329	Op330	Op331	Op332	Op333	Op334	Op335	Op336	Op337	Op338	Op339	Op340	Op341	Op342	Op343	Op344	Op345	Op346	Op347	Op348	Op349	Op350	Op351	Op352	Op353	Op354	Op355	Op356	Op357	Op358	Op359	Op360	Op361	Op362	Op363	Op364	Op365	Op366	Op367	Op368	Op369	Op370	Op371	Op372	Op373	Op374	Op375	Op376	Op377	Op378	Op379	Op380	Op381	Op382	Op383	Op384	Op385	Op386	Op387	Op388	Op389	Op390	Op391	Op392	Op393	Op394	Op395	Op396	Op397	Op398	Op399	Op400	Op401	Op402	Op403	Op404	Op405	Op406	Op407	Op408	Op409	Op410	Op411	Op412	Op413	Op414	Op415	Op416	Op417	Op418	Op419
----	----	-----	-----	-----	-----	-----	-----	-----	-----	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------	-------



66	0C	B9	08	A9	28	00152	MOV C3	LEX_STRING_DESC, @LEX_STRING_DESC+4, -	
		63	18	AA	90	00158	MOV B	P.AAT, (POINTER)	
	08	AE		56	D0	0015C	MOVL	NUM BUF, NUMBER_DESC+4	
			E0	A9	9F	00160	PUSHAB	DUMMY	
			04	AE	9F	00163	PUSHAB	NUMBER	
			0C	AE	9F	00166	PUSHAB	NUMBER_DESC	
00000000G		00		03	FB	00169	CALLS	#3, DBG\$NSAVE_DECIMAL_INTEGER	
		3A		50	E9	00170	BLBC	R0, 13\$	
		50	E8	A9	D0	00173	MOVL	PATHNAME_DESC, R0	
	02	A0	F0	A9	90	00177	MOV B	NAME_INDEX, 2(R0)	
	04	A0		6E	D0	0017C	MOVL	NUMBER, 4(R0)	
D0	A9	08		08	28	00180	MOV C3	#8, LEX_STRING_DESC, LAST_TOKEN_DESC	2015
		50	E4	A9	D0	00186	MOVL	INPUT_DESC, R0	
51	0C	A9	04	A0	C3	0018A	SUBL3	4(R0), LEX_STRING_DESC+4, R1	
		52	08	A9	3C	00190	MOVZWL	LEX_STRING_DESC, R2	
		51		52	C0	00194	ADDL2	R2, R1	
		60		51	A2	00197	SUBW2	R1, (R0)	
	04	A0	0C	B9	42	9E	0019A	MOVAB	@LEX_STRING_DESC+4[R2], 4(R0)
	DC	A9		69	D0	001A0	MOVL	TOKEN, LAST_TOKEN	
		60		0B	11	001A4	BRB	14\$	2007
		04		58	B0	001A6	MOVW	LENGTH, (R0)	2023
		A0		57	D0	001A9	MOVL	POINTER, 4(R0)	
		50		04	D0	001AD	MOVL	#4, R0	2024
				04	001B0		RET		
		50		01	D0	001B1	MOVL	#1, R0	2035
				04	001B4		RET		2037

; Routine Size: 437 bytes, Routine Base: DBG\$CODE + 0F42

; 1925 2038 1

```
1927 2039 1 GLOBAL ROUTINE CHECK_PATHNAME : NOVALUE =
1928 2040 1
1929 2041 1 ++
1930 2042 1 FUNCTIONAL DESCRIPTION:
1931 2043 1
1932 2044 1 This routine examines a completed pathname descriptor and classifies its
1933 2045 1 type by setting the value state to:
1934 2046 1
1935 2047 1 dbg$_reg : register reference (item count is 0)
1936 2048 1
1937 2049 1 dbg$_line : line number reference (not a data item)
1938 2050 1
1939 2051 1 dbg$_label : numeric label reference (not a data item)
1940 2052 1
1941 2053 1 dbg$_pn : data or lexical item reference
1942 2054 1
1943 2055 1 FORMAL PARAMETERS:
1944 2056 1
1945 2057 1 NONE
1946 2058 1
1947 2059 1 IMPLICIT INPUTS:
1948 2060 1
1949 2061 1 The pathname descriptor constructed by parse_pathname.
1950 2062 1
1951 2063 1 IMPLICIT OUTPUTS:
1952 2064 1
1953 2065 1 NONE
1954 2066 1
1955 2067 1 ROUTINE VALUE:
1956 2068 1
1957 2069 1 NOVALUE
1958 2070 1
1959 2071 1 COMPLETION CODES:
1960 2072 1
1961 2073 1 NONE
1962 2074 1
1963 2075 1 SIDE EFFECTS:
1964 2076 1
1965 2077 1 The value state is set according to the type of pathname descriptor examined.
1966 2078 1
1967 2079 1 --
1968 2080 2 BEGIN
1969 2081 2
1970 2082 2 BIND
1971 2083 2 LINE_STG = UPLIT BYTE ('%LINE'),
1972 2084 2 LABEL_STG = UPLIT BYTE ('%LABEL');
1973 2085 2
1974 2086 2 LOCAL
1975 2087 2 NEW_STRING : REF VECTOR [,BYTE],
1976 2088 2 STRING : REF VECTOR [,BYTE]; ! String vector
1977 2089 2
1978 2090 2
1979 2091 2 string = .name_vect [.pathname_desc [pth$b_totcnt] - 1];
1980 2092 2
1981 2093 2
1982 2094 2 ! If language is C, then copy and upcase the string.
1983 2095 2
```



```

IF .dbg$gb_language EQL dbg$k_c
THEN
  BEGIN
    new_string = dbg$get_tempmem((.string[0]/4)+1);
    ch$move(.string[0]+1, .string, .new_string);
    INCR i FROM 1 TO .new_string[0] DO
      IF .new_string[i] GEQ 'a' AND .new_string[i] LEQ 'z'
      THEN
        new_string[i] = .new_string[i] - ('a' - 'A');
    string = .new_string;
  END;

! Set the value state by examining the completed pathname descriptor
!
SELECTONE true
OF
SET
  [.pathname_desc [pth$b_totcnt] EQL 0] : value_state = dbg$k_reg;
  [ch$find_sub (.string [0], string [1], 5, line_stg) NEQA 0] : value_state = dbg$k_line;
  [ch$find_sub (.string [0], string [1], 6, label_stg) NEQA 0] : value_state = dbg$k_label;
  [OTHERWISE] : value_state = dbg$k_pn;
TES;

RETURN;

END;                ! End of check_pathname

```

										.PSECT		DBG\$PLIT,NOWRT, SHR, PIC,0						
4C	45	4E	49	4C	25	00019	P.AAJ:	.ASCII	\%LINE\	:								
4C	45	42	41	4C	25	0001E	P.AAK:	.ASCII	\%LABEL\	:								
										LINE_STG=		P.AAJ						
										LABEL_STG=		P.AAK						
										.PSECT		DBG\$CODE,NOWRT, SHR, PIC,0						
										.ENTRY		CHECK_PATHNAME, Save R2,R3,R4,R5,R6,R7,R8		:	2039			
58	00000000'	EF	9E	00002						MOVAB	VALUE_STATE, R8	:						
50	F4	B8	9A	00009						MOVZBL	@PATHNAME_DESC, R0	:	2091					
50	F8	B840	DE	0000D						MOVAL	@NAME_VECT[R0], R0	:						
56	FC	A0	D0	00012						MOVL	-4(R0), STRING	:						
07	00000000G	00	91	00016						CMPB	DBG\$GB_LANGUAGE, #7	:	2096					
											3C	12	0001D			:		
50		66	9A	0001F						BNEQ	3\$	:						
50		04	C6	00022						MOVZBL	(STRING), R0	:	2099					
											01	A0	9F	00025			:	
00000000G	00	01	FB	00028						DIVL2	#4, R0	:						
														1(R0)			:	
														#1, DBG\$GET_TEMPMEM			:	

		57	50	D0	0002F	MOVL	R0, NEW_STRING	:			
		50	66	9A	00032	MOVZBL	(STRING), R0	:	2100		
			50	D6	00035	INCL	R0	:			
67		66	50	28	00037	MOVCL	R0, (STRING), (NEW_STRING)	:			
		51	67	9A	0003B	MOVZBL	(NEW_STRING), R1	:	2101		
			50	D4	0003E	CLRL	I	:			
			12	11	00040	BRB	2\$	:			
	61	8F	6047	91	00042	1\$:	CMPB	(I)[NEW_STRING], #97	:	2102	
			0B	1F	00047		BLSSU	2\$	:		
	7A	8F	6047	91	00049		CMPB	(I)[NEW_STRING], #122	:		
			04	1A	0004E		BGTRU	2\$	:		
		6047	20	82	00050		SUBB2	#32, (I)[NEW_STRING]	:	2104	
EA		50	51	F3	00054	2\$:	AOBLEQ	R1, I, 1\$	:	2102	
		56	57	D0	00058		MOVL	NEW_STRING, STRING	:	2105	
			F4	B8	95	0005B	3\$:	TSTB	@PATHNAME_DESC	:	2115
			04	12	0005E		BNEQ	4\$	:		
		68	01	D0	00060		MOVL	#1, VALUE_STATE	:		
			04	00	00063		RET		:		
		50	66	9A	00064	4\$:	MOVZBL	(STRING), R0	:	2117	
01	A6		05	39	00067		MATCHC	#5, LINE_STG, R0, 1(STRING)	:		
		EF	03	13	00071		BEQL	5\$	:		
		53	05	D0	00073		MOVL	#5, R3	:		
		53	05	C2	00076	5\$:	SUBL2	#5, R3	:		
			04	13	00079		BEQL	6\$	:		
		68	02	D0	0007B		MOVL	#2, VALUE_STATE	:		
			04	00	0007E		RET		:		
		50	66	9A	0007F	6\$:	MOVZBL	(STRING), R0	:	2119	
01	A6		06	39	00082		MATCHC	#6, LABEL_STG, R0, 1(STRING)	:		
		EF	03	13	0008C		BEQL	7\$	:		
		53	06	D0	0008E		MOVL	#6, R3	:		
		53	06	C2	00091	7\$:	SUBL2	#6, R3	:		
			04	13	00094		BEQL	8\$	:		
		68	03	D0	00096		MOVL	#3, VALUE_STATE	:		
			04	00	00099		RET		:		
			68	D4	0009A	8\$:	CLRL	VALUE_STATE	:	2121	
			04	00	0009C		RET		:	2127	

; Routine Size: 157 bytes, Routine Base: DBG\$CODE + 10F7

; 2016 2128 1



```
2018 2129 1 GLOBAL ROUTINE DBG$NPATHDESC_TO_CS (PATHNAME_DESC, COUNTED_STRING) : NOVALUE =
2019 2130 1
2020 2131 1 ++
2021 2132 1 FUNCTIONAL DESCRIPTION:
2022 2133 1
2023 2134 1 This routine accepts a pathname descriptor and translates the contents of
2024 2135 1 the descriptor into a printable form. That is, the names and optional
2025 2136 1 invocation number contained within the pathname descriptor are formatted
2026 2137 1 into one long counted string.
2027 2138 1
2028 2139 1 This routine will produce the translation for any pathname descriptor which
2029 2140 1 describes a legal scope including '\' and numeric scopes.
2030 2141 1
2031 2142 1 Pathnames in which the first two names are the same are modified to
2032 2143 1 output the name only once (situations where routine and module names
2033 2144 1 are the same).
2034 2145 1
2035 2146 1 FORMAL PARAMETERS:
2036 2147 1
2037 2148 1 PN_DESC - A longword containing the address of a pathnaem
2038 2149 1 descriptor
2039 2150 1
2040 2151 1 COUNTED_STRING - The address of a longword to contain the address
2041 2152 1 of a counted string representing the translation
2042 2153 1 of the contents of the pathname descriptor
2043 2154 1
2044 2155 1 IMPLICIT INPUTS:
2045 2156 1
2046 2157 1 NONE
2047 2158 1
2048 2159 1 IMPLICIT OUTPUTS:
2049 2160 1
2050 2161 1 The translated pathname string
2051 2162 1
2052 2163 1 ROUTINE VALUE:
2053 2164 1
2054 2165 1 NOVALUE
2055 2166 1
2056 2167 1 COMPLETION CODES:
2057 2168 1
2058 2169 1 NONE
2059 2170 1
2060 2171 1 SIDE EFFECTS:
2061 2172 1
2062 2173 1 This routine will produce a SIGNAL for certain circumstances.
2063 2174 1
2064 2175 1 --
2065 2176 2 BEGIN
2066 2177 2
2067 2178 2 MAP
2068 2179 2 PATHNAME_DESC : REF pth$pathname;
2069 2180 2
2070 2181 2 LOCAL
2071 2182 2 SAVE_STRING, ! Pointer to original string
2072 2183 2 PATH_STRING : REF VECTOR [,BYTE], ! Result buffer
2073 2184 2 NAME_VECT : REF VECTOR, ! Vector of pointers to name strings
2074 2185 2 NAME : REF VECTOR [,BYTE], ! Name counted string
```

```

2075      INDEX      : 2186      INDEX      : Index into name_vect
2076      SOURCE_DESC : 2187      SOURCE_DESC : Source descriptor
2077      TARGET_DESC : 2188      TARGET_DESC : Target descriptor
2078      RESULT_LENGTH : 2189      RESULT_LENGTH : Length of string after FA0ing
2079      NEXT_CHAR,    : 2190      NEXT_CHAR,    : Pointer into result string
2080      SIZE;         : 2191      SIZE;         : Number of bytes needed for result buffer
2081
2082      save_string = 0;
2083
2084      ! Line up the name vector
2085
2086      name_vect = pathname_desc [pth$a_pathvector];
2087
2088
2089      ! Look for an invocation number. If there is one, go ahead and add the number
2090      ! to the correct name string. We save the original name string so that we may restore it.
2091
2092      IF .pathname_desc [pth$b_locinvoc] NEQ 0
2093      THEN
2094      BEGIN
2095          ! Recover the name string
2096
2097          name = .name_vect [.pathname_desc [pth$b_locinvoc] - 1];
2098          save_string = .name;
2099
2100
2101          ! Allocate enough storage to concatenate the number to the string
2102
2103          path_string = dbg$get_tempmem((.name [0] + 24) / %UPVAL);
2104
2105
2106          ! Copy the name string
2107
2108          IF .name [0] NEQ 0
2109          THEN
2110          BEGIN
2111              next_char = ch$move (.name [0], name [1], path_string [1]);
2112              source_desc [dsc$a_pointer] = UPLIT BYTE ('!UL');
2113              source_desc [dsc$w_length] = 4;
2114          END
2115          ELSE
2116          BEGIN
2117              next_char = path_string [1];
2118              source_desc [dsc$a_pointer] = UPLIT BYTE ('!UL');
2119              source_desc [dsc$w_length] = 3;
2120          END;
2121
2122
2123          ! Append the invocation number
2124
2125          target_desc [dsc$a_pointer] = .next_char;
2126          target_desc [dsc$w_length] = 23;
2127
2128          sys$fao (source_desc, result_length, target_desc, .pathname_desc [pth$l_invocnum]);
2129
2130
2131
```



```
2132      2243      ! Update the copie's length
2133      2244      !
2134      2245      path_string [0] = .name [0] + .result_length;
2135      2246
2136      2247
2137      2248      ! Point to the copy
2138      2249      !
2139      2250      name_vect [.pathname_desc [pth$b_locinvoc] - 1] = .path_string;
2140      2251      END;
2141      2252
2142      2253
2143      2254      ! Figure out how much space will be needed to hold the entire string
2144      2255      !
2145      2256      size = 0;
2146      2257      INCR index FROM 0 TO .pathname_desc [pth$b_totcnt] - 1
2147      2258      DO
2148      2259      BEGIN
2149      2260      name = .name_vect [.index];
2150      2261      size = .size + .name [0] + 1;    ! One for '\'
2151      2262      END;
2152      2263
2153      2264
2154      2265      ! Allocate enough storage to hold the string plus one byte for the length
2155      2266      !
2156      2267      path_string = dbg$get_tempmem((.size / %UPVAL) + 2);
2157      2268
2158      2269
2159      2270      ! Now we're ready to append all the name strings into one string. First
2160      2271      ! check for the special case of the global scope, '\'.
2161      2272      !
2162      2273      name = .name_vect [0];
2163      2274      IF .name [0] EQL 0 AND .pathname_desc [pth$b_locinvoc] EQL 0
2164      2275      THEN
2165      2276      BEGIN
2166      2277
2167      2278      ! Global scope or global reference
2168      2279      !
2169      2280      ch$move (1, UPLIT BYTE ('\'), path_string [1]);
2170      2281      result_length = 1;
2171      2282      IF .pathname_desc [pth$b_totcnt] GTR 1
2172      2283      THEN
2173      2284      BEGIN
2174      2285      name = .name_vect [1];
2175      2286      ch$move (.name [0], name [1], path_string [2]);
2176      2287      result_length = .result_length + .name [0];
2177      2288      END;
2178      2289      END
2179      2290      ELSE
2180      2291      BEGIN
2181      2292      LOCAL
2182      2293      I,
2183      2294      NAME_1      : REF VECTOR [,BYTE],
2184      2295      NAME_2      : REF VECTOR [,BYTE];
2185      2296
2186      2297      ! Loop, adding all the name strings.
2187      2298      !
2188      2299      result_length = 0;
```

```
2189      2300      next_char = path_string [1];
2190      2301
2191      2302
2192      2303      ! We do not want to output the same name twice. Check to see if the
2193      2304      ! first name and the second name are the same. If they are, skip over
2194      2305      ! the first name.
2195      2306
2196      2307      i = 0;
2197      2308      IF .pathname_desc [pth$b_totcnt] GEQ 2
2198      2309      THEN
2199      2310      BEGIN
2200      2311      name_1 = .name_vect [0];
2201      2312      name_2 = .name_vect [1];
2202      2313      IF ch$seq1 (.name_1 [0], name_1 [1], .name_2 [0], name_2 [1])
2203      2314      THEN
2204      2315      i = 1;
2205      2316      END;
2206      2317
2207      2318      INCR index FROM .i TO .pathname_desc [pth$b_totcnt] - 1
2208      2319      DO
2209      2320      BEGIN
2210      2321      name = .name_vect [.index];
2211      2322      next_char = ch$move (.name [0], name [1], .next_char);
2212      2323      result_length = .result_length + .name [0];
2213      2324
2214      2325
2215      2326      ! If there is another name string, we add a '\'
2216      2327
2217      2328      IF .index LSS .pathname_desc [pth$b_totcnt] - 1
2218      2329      THEN
2219      2330      BEGIN
2220      2331      IF .index LSS .pathname_desc [pth$b_pathcnt] - 1
2221      2332      THEN
2222      2333      next_char = ch$move (1, UPLIT BYTE ('\'), .next_char)
2223      2334      ELSE
2224      2335      next_char = ch$move (1, UPLIT BYTE ('.'), .next_char);
2225      2336      result_length = .result_length + 1;
2226      2337      END;
2227      2338      END;
2228      2339      END;
2229      2340
2230      2341
2231      2342      ! Fill in the count byte. Check for overflow.
2232      2343
2233      2344      path_string [0] = (IF .result_length GTR 255 THEN 255 ELSE .result_length);
2234      2345
2235      2346
2236      2347      ! Restore the saved string if there is one.
2237      2348
2238      2349      IF .save_string NEQA 0
2239      2350      THEN
2240      2351      name_vect [.pathname_desc [pth$b_locinvoc] - 1] = .save_string;
2241      2352
2242      2353
2243      2354      ! Return the counted string
2244      2355
2245      2356      .counted_string = .path_string;
```



: 2246  
: 2247  
: 2248  
: 2249

2357 2  
2358 2  
2359 2  
2360 1

RETURN;  
END;

! End of DBG\$NPATHDESC\_TO\_CS

.PSECT DBG\$PLIT,NOWRT, SHR, PIC,0

4C 55 21 20 00024 P.AAL: .ASCII \!UL\  
4C 55 21 21 00028 P.AAM: .ASCII \!UL\  
5C 0002B P.AAN: .ASCII <92>  
5C 0002C P.AAO: .ASCII <92>  
2E 0002D P.AAP: .ASCII \.

.PSECT DBG\$CODE,NOWRT, SHR, PIC,0

				OFFC 00000	.ENTRY	DBG\$NPATHDESC_TO_CS, Save R2,R3,R4,R5,R6,-	2129
						R7,R8,R9,R10,R11	
		5E	20	C2 00002	SUBL2	#32, SP	
			7E	D4 00005	CLRL	SAVE STRING	2193
		5A	04	AC D0 00007	MOVL	PATHNAME_DESC, R10	2197
		57	08	AA 9E 0000B	MOVAB	8(R10), NAME_VECT	
			02	AA 95 0000F	TSTB	2(R10)	2203
				76 13 00012	BEQL	3\$	
		56	02	AA 9A 00014	MOVZBL	2(R10), R6	2209
		58	FC	A746 D0 00018	MOVL	-4(NAME_VECT)[R6], NAME	
		6E		58 D0 0001D	MOVL	NAME, SAVE_STRING	2210
		50		68 9A 00020	MOVZBL	(NAME), R0	2215
		50		18 C0 00023	ADDL2	#24, R0	
7E	00000000G	50		04 C7 00026	DIVL3	#4, R0, -(SP)	
		00		01 FB 0002A	CALLS	#1, DBG\$GET_TEMPMEM	
		59		50 D0 00031	MOVL	R0, PATH_STRING	
				68 95 00034	TSTB	(NAME)	2220
				1B 13 00036	BEQL	1\$	
		50		68 9A 00038	MOVZBL	(NAME), R0	2223
01	A9	01		50 28 0003B	MOVC3	R0, 1(NAME), 1(PATH_STRING)	
		04		53 D0 00041	MOVL	R3, NEXT_CHAR	
		1C		EF 9E 00045	MOVAB	P.AAL, SOURCE_DESC+4	2224
		18		04 B0 0004D	MOVW	#4, SOURCE_DESC	2225
				11 11 00051	BRB	2\$	2220
		04	01	A9 9E 00053	MOVAB	1(PATH_STRING), NEXT_CHAR	2229
		1C		EF 9E 00058	MOVAB	P.AAM, SOURCE_DESC+4	2230
		18		03 B0 00060	MOVW	#3, SOURCE_DESC	2231
		10	04	AE D0 00064	MOVL	NEXT_CHAR, TARGET_DESC+4	2237
		0C		17 B0 00069	MOVW	#23, TARGET_DESC	2238
			04	AA DD 0006D	PUSHL	4(R10)	2240
			10	AE 9F 00070	PUSHAB	TARGET_DESC	
			10	AE 9F 00073	PUSHAB	RESULT_LENGTH	
			24	AE 9F 00076	PUSHAB	SOURCE_DESC	
	00000000G	9F		04 FB 00079	CALLS	#4, @#SYSSFAO	
69		68	08	AE 81 00080	ADDB3	RESULT_LENGTH, (NAME), (PATH_STRING)	2245
	FC A746			59 D0 00085	MOVL	PATH_STRING, -4(NAME_VECT)[R6]	2250
				50 D4 0008A	CLRL	SIZE	2256
		5B		6A 9A 0008C	MOVZBL	(R10), R11	2257



51	01	CE	0008F	MNEGL	#1, INDEX	2260		
	OC	11	00092	BRB	5\$			
58	6741	D0	00094	4\$:	MOVL	(NAME_VECT)[INDEX], NAME		
52	68	9A	00098	MOVZBL	(NAME), R2	2261		
50	01 A240	9E	0009B	MOVAB	1(R2)[SIZE], SIZE			
51	5B	F2	000A0	5\$:	A0BLSS	R11, INDEX, 4\$		
50	04	C6	000A4	DIVL2	#4, R0	2257		
	02	A0	9F	000A7	PUSHAB	2(R0)		
00	01	FB	000AA	CALLS	#1, DBG\$GET_TEMPMEM			
59	50	D0	000B1	MOVL	R0, PATH_STRING			
58	67	D0	000B4	MOVL	(NAME_VECT), NAME	2273		
	68	95	000B7	TSTB	(NAME)	2274		
	2C	12	000B9	BNEQ	7\$			
	02	AA	95	000BB	TSTB	2(R10)		
	27	12	000BE	BNEQ	7\$			
01	A9	00000000'	EF	90	000C0	MOVAB	P.AAN, 1(PATH_STRING)	2280
08	AE		01	B0	000C8	MOVW	#1, RESULT_LENGTH	2281
	01		6A	91	000CC	CMPB	(R10), #1	2282
			14	1B	000CF	BLEQU	6\$	
58	04		A7	D0	000D1	MOVL	4(NAME_VECT), NAME	2285
50			68	9A	000D5	MOVZBL	(NAME), R0	2286
01	A8		50	28	000D8	MOV3	R0, 1(NAME), 2(PATH_STRING)	
50			68	9A	000DE	MOVZBL	(NAME), R0	2287
08	AE		50	A0	000E1	ADDW2	R0, RESULT_LENGTH	
			75	11	000E5	6\$:	BRB	13\$
			AE	B4	000E7	7\$:	CLRW	RESULT_LENGTH
04	AE		01	A9	9E	000EA	MOVAB	1(PATH_STRING), NEXT_CHAR
			56	D4	000EF	CLRL	I	2300
02			6A	91	000F1	CMPB	(R10), #2	2307
			1A	1F	000F4	BLSSU	8\$	2308
51			67	D0	000F6	MOVL	(NAME_VECT), NAME_1	2311
50	04		A7	D0	000F9	MOVL	4(NAME_VECT), NAME_2	2312
53			61	9A	000FD	MOVZBL	(NAME_1), R3	2313
52			60	9A	00100	MOVZBL	(NAME_2), R2	
52	00	01	A1	2D	00103	CMPC5	R3, 1(NAME_1), #0, R2, 1(NAME_2)	
		01	A0		00109			
			03	12	0010B	BNEQ	8\$	
56			01	D0	0010D	MOVL	#1, I	2315
			56	D7	00110	8\$:	DECL	INDEX
			44	11	00112	BRB	12\$	2318
58			6746	D0	00114	9\$:	MOVL	(NAME_VECT)[INDEX], NAME
50			68	9A	00118	MOVZBL	(NAME), R0	2321
01	A8		50	28	0011B	MOV3	R0, 1(NAME), @NEXT_CHAR	2322
04	AE		53	D0	00121	MOVL	R3, NEXT_CHAR	
50			68	9A	00125	MOVZBL	(NAME), R0	2323
08	AE		50	A0	00128	ADDW2	R0, RESULT_LENGTH	
50		FF	AB	9E	0012C	MOVAB	-1(R11), R0	2328
50			56	D1	00130	CMPL	INDEX, R0	
			23	18	00133	BGEQ	12\$	
50	01		AA	9A	00135	MOVZBL	1(R10), R0	2331
			50	D7	00139	DECL	R0	
50			56	D1	0013B	CMPL	INDEX, R0	
			0A	18	0013E	BGEQ	10\$	
04	BE	00000000'	EF	90	00140	MOVAB	P.AAO, @NEXT_CHAR	2333
			08	11	00148	BRB	11\$	
04	BE	00000000'	EF	90	0014A	10\$:	MOVAB	P.AAP, @NEXT_CHAR
		04	AE	D6	00152	11\$:	INCL	NEXT_CHAR



B8		56	08	AE	B6	00155		INCW	RESULT_LENGTH	:	2336
	00FF	8F		5B	F2	00158	12\$:	AOBLSS	R11, INDEX, 9\$	:	2318
			08	AE	B1	0015C	13\$:	CMPL	RESULT_LENGTH, #255	:	2344
		50		06	1B	00162		BLEQU	14\$	:	
			FF	8F	9A	00164		MOVZBL	#255, R0	:	
		50		04	11	00168		BRB	15\$	:	
		69	08	AE	3C	0016A	14\$:	MOVZWL	RESULT_LENGTH, R0	:	
				50	90	0016E	15\$:	MOVB	R0, (PATH_STRING)	:	
				6E	D5	00171		TSTL	SAVE_STRING	:	2349
		50		09	13	00173		BEQL	16\$	:	
	FC A740		02	AA	9A	00175		MOVZBL	2(R10), R0	:	2351
	08 BC			6E	D0	00179		MOVL	SAVE_STRING, -4(NAME_VECT)[R0]	:	
				59	D0	0017E	16\$:	MOVL	PATH_STRING, @COUNTED_STRING	:	2356
				04	00182			RET		:	2360

; Routine Size: 387 bytes, Routine Base: DBG\$CODE + 1194

; 2250 2361 1

```
2252 2362 1 ROUTINE SCOPE_SCANNER (INPUT_DESC, LEX_DESC, TOKEN) : NOVALUE =
2253 2363 1
2254 2364 1 ++
2255 2365 1 FUNCTIONAL DESCRIPTION:
2256 2366 1
2257 2367 1 Lexical scanner for the parsing of scopes. This routine supplies
2258 2368 1 tokens to the pathname parser when a scope is to be parsed. It plays
2259 2369 1 the part of a language specific lexical scanner and its address is
2260 2370 1 supplied to the pathname parser by dbg$npars_scope_list.
2261 2371 1
2262 2372 1 The tokens returned by this routine are limited to:
2263 2373 1
2264 2374 1 dbg$k_tok_null, dbg$k_tok_inval, dbg$k_tok_line, dbg$k_tok_label,
2265 2375 1 dbg$k_tok_int, dbg$k_tok_id, dbg$k_tok_dot, and dbg$k_tok_bs.
2266 2376 1
2267 2377 1 Note that unlike the actual language specific scanners, this routine does
2268 2378 1 not return a token for %register since these are invalid in a scope.
2269 2379 1
2270 2380 1 The input line is NOT updated after a token is recognized. The caller
2271 2381 1 of this routine is responsible for updating the input line by
2272 2382 1 using the information in the lexical string descriptor.
2273 2383 1
2274 2384 1 The input line is assumed to be terminated with a <CR>.
2275 2385 1
2276 2386 1 FORMAL PARAMETERS:
2277 2387 1
2278 2388 1 INPUT_DESC - A longword containing the address of a standard
2279 2389 1 ascii string descriptor representing the input line
2280 2390 1
2281 2391 1 LEX_DESC - A longword containing the address of a standard
2282 2392 1 ascii string descriptor. The length and a pointer
2283 2393 1 fields of this descriptor are filled in to reflect
2284 2394 1 the portion of the input which represents the token
2285 2395 1 recognized.
2286 2396 1
2287 2397 1 TOKEN - The address of a longword to contain the value
2288 2398 1 of the token recognized
2289 2399 1
2290 2400 1 IMPLICIT INPUTS:
2291 2401 1
2292 2402 1 NONE
2293 2403 1
2294 2404 1 IMPLICIT OUTPUTS:
2295 2405 1
2296 2406 1 Token value is returned and the lexical string descriptor is updated.
2297 2407 1
2298 2408 1 ROUTINE VALUE:
2299 2409 1
2300 2410 1 NOVALUE
2301 2411 1
2302 2412 1 COMPLETION CODES:
2303 2413 1
2304 2414 1 NONE
2305 2415 1
2306 2416 1 SIDE EFFECTS:
2307 2417 1
2308 2418 1 NONE
```





```
: 2366      2476      3
: 2367      2477      3
: 2368      2478      3
: 2369      2479      3
: 2370      2480      3
: 2371      2481      3
: 2372      2482      3
: 2373      2483      4
: 2374      2484      4
: 2375      2485      4
: 2376      2486      4
: 2377      2487      4
: 2378      2488      4
: 2379      2489      4
: 2380      2490      4
: 2381      2491      4
: 2382      2492      4
: 2383      2493      4
: 2384      2494      4
: 2385      2495      4
: 2386      2496      4
: 2387      2497      4
: 2388      2498      4
: 2389      2499      4
: 2390      2500      4
: 2391      2501      4
: 2392      2502      3
: 2393      2503      4
: 2394      2504      4
: 2395      2505      4
: 2396      2506      4
: 2397      2507      4
: 2398      2508      3
: 2399      2509      3
: 2400      2510      3
: 2401      2511      3
: 2402      2512      3
: 2403      2513      3
: 2404      2514      3
: 2405      2515      3
: 2406      2516      3
: 2407      2517      3
: 2408      2518      3
: 2409      2519      3
: 2410      2520      3
: 2411      2521      3
: 2412      2522      3
: 2413      2523      3
: 2414      2524      3
: 2415      2525      3
: 2416      2526      3
: 2417      2527      3
: 2418      2528      3
: 2419      2529      3
: 2420      2530      4
: 2421      2531      4
: 2422      2532      3
```

```
BEGIN
LOCAL
  STRING_DESC : dbg$stg_desc;

IF .dbg$gb_language EQL dbg$k_c
THEN
  BEGIN
    ! Copy and upcase the string.
    !
    string = ch$plus (.pointer, 1);
    length = .input_desc [dsc$w_length] -
      (.pointer + 1 - .input_desc [dsc$a_pointer]);
    new_string = dbg$get_tempmem((length+3)/4);
    ch$move(length, .string, .new_string);
    INCR i FROM 0 TO .length - 1 DO
      IF .new_string[i] GEQ 'a' AND .new_string[i] LEQ 'z'
      THEN
        new_string[i] = .new_string[i] - ('a' - 'A');
    string_desc [dsc$w_length] = .length;
    string_desc [dsc$a_pointer] = .new_string;
  END

  ! All other languages.
ELSE
  BEGIN
    string_desc [dsc$a_pointer] = ch$plus (.pointer, 1);
    string_desc [dsc$w_length] = .input_desc [dsc$w_length] -
      (.pointer + 1 - .input_desc [dsc$a_pointer]);
  END;

  SELECTONE true
  OF
  SET

    [dbg$nmatch (string_desc, UPLIT BYTE (%ASCIC 'LINE'), 2)] :
      .token = dbg$k_tok_line;

    [dbg$nmatch (string_desc, UPLIT BYTE (%ASCIC 'LABEL'), 2)] :
      .token = dbg$k_tok_label;

    [dbg$nmatch (string_desc, UPLIT BYTE (%ASCIC 'NAME'), 1)] :
      .token = dbg$k_tok_qname;

    [OTHERWISE] :
      .token = dbg$k_tok_inval;

  TES;

  IF .dbg$gb_language EQL dbg$k_c
  THEN
    token_end = .pointer +
      (.string_desc[dsc$a_pointer] - .new_string)

  ELSE
```



```
: 2423      2533      3
: 2424      2534      3
: 2425      2535      3
: 2426      2536      3
: 2427      2537      3
: 2428      2538      3
: 2429      2539      3
: 2430      2540      3
: 2431      2541      3
: 2432      2542      3
: 2433      2543      3
: 2434      2544      3
: 2435      2545      3
: 2436      2546      3
: 2437      2547      3
: 2438      2548      3
: 2439      2549      3
: 2440      2550      3
: 2441      2551      3
: 2442      2552      3
: 2443      2553      3
: 2444      2554      3
: 2445      2555      3
: 2446      2556      3
: 2447      2557      3
: 2448      2558      3
: 2449      2559      3
: 2450      2560      3
: 2451      2561      3
: 2452      2562      3
: 2453      2563      3
: 2454      2564      3
: 2455      2565      3
: 2456      2566      3
: 2457      2567      3
: 2458      2568      3
: 2459      2569      3
: 2460      2570      3
: 2461      2571      3
: 2462      2572      3
: 2463      2573      3
: 2464      2574      3
: 2465      2575      3
: 2466      2576      3
: 2467      2577      3
: 2468      2578      3
: 2469      2579      3
: 2470      2580      1

      token_end = .string_desc [dsc$a_pointer];
      END;

[.char GEQ '0' AND .char LEQ '9'] :      ! Integer
      BEGIN
      WHILE .char GEQ '0' AND .char LEQ '9' DO char = ch$a_rchar (pointer);

      token_end = .pointer;
      .token = dbg$k_tok_int;
      END;

[(.char GEQ 'A' AND .char LEQ 'Z') OR
(.char GEQ 'a' AND .char LEQ 'z')] :      ! ID
      BEGIN
      WHILE .char NEQ ','
      AND
      .char NEQ '\'
      AND
      .char NEQ ' '
      AND
      .char NEQ dbg$k_car_return
      DO
      BEGIN
      pointer = ch$plus (.pointer, 1);
      char = ch$rchar (.pointer);
      END;

      token_end = .pointer;
      .token = dbg$k_tok_id;
      END;

[OTHERWISE] :
      BEGIN
      .token = dbg$k_tok_inval;
      END;

      TES;

! Now fill in the lexical string descriptor
lex_string_desc [dsc$a_pointer] = .token_start;
lex_string_desc [dsc$w_length] = .token_end - .token_start;

      RETURN;

      END;      ! End of SCOPE_SCANNER
```

.PSECT DBG\$PLIT, NOWRT, SHR, PIC, 0

```
4C 45 4E 49 4C 04 0002E P.AAQ: .ASCII <4>\LINE\
4C 45 42 41 4C 05 00033 P.AAR: .ASCII <5>\LABEL\
4C 45 4D 41 4E 04 00039 P.AAS: .ASCII <4>\NAME\
```

...

				.PSECT		DBG\$CODE,NOWRT, SHR, PIC,0			
				OFFC 00000		SCOPE_SCANNER:			
	5E		0C	C2	00002	.WORD	Save R2,R3,R4,R5,R6,R7,R8,R9,R10,R11	:	2362
	50	04	AC	D0	00005	SUBL2	#12, SP	:	2437
	56	04	A0	D0	00009	MOVL	INPUT_DESC, R0	:	2442
	59		66	90	0000D	MOVL	4(R0), POINTER	:	2443
	20		59	91	00010	MOVB	(POINTER), CHAR	:	
			04	12	00013	CMPB	CHAR, #32	:	
			56	D6	00015	BNEQ	2\$	:	
			F4	11	00017	INCL	POINTER	:	
	5B		56	D0	00019	BRB	1\$	:	
	0D		59	91	0001C	MOVL	POINTER, TOKEN_START	:	2448
			08	12	0001F	CMPB	CHAR, #13	:	2457
	5A		5B	D0	00021	BNEQ	3\$	:	
		0C	BC	D4	00024	MOVL	TOKEN_START, TOKEN_END	:	2459
			1D	11	00027	CLRL	@TOKEN	:	2460
5C	8F		59	91	00029	BRB	5\$	:	2453
			0A	12	0002D	CMPB	CHAR, #92	:	2463
	5A	01	AB	9E	0002F	BNEQ	4\$	:	
0C	BC		04	D0	00033	MOVAB	1(R11), TOKEN_END	:	2465
			0D	11	00037	MOVL	#4, @TOKEN	:	2466
	2E		59	91	00039	BRB	5\$	:	2453
			0B	12	0003C	CMPB	CHAR, #46	:	2469
	5A	01	AB	9E	0003E	BNEQ	6\$	:	
0C	BC		07	D0	00042	MOVAB	1(R11), TOKEN_END	:	2471
		017B	31	00046	MOVL	#7, @TOKEN	:	2472	
	25		59	91	00049	BRW	31\$	:	2453
			03	13	0004C	CMPB	CHAR, #37	:	2475
		00D7	31	0004E	BEQL	7\$	:		
	51	01	A6	9E	00051	BRW	17\$	:	
	07	00000000G	00	91	00055	MOVAB	1(R6), R1	:	2487
			48	12	0005C	CMPB	DBG\$GB_LANGUAGE, #7	:	2481
	53		51	D0	0005E	BNEQ	10\$	:	
52	04		51	C3	00061	MOVL	R1, STRING	:	2487
			60	3C	00066	SUBL3	R1, 4(R0), R2	:	2489
			52	C0	00069	MOVZWL	(R0), LENGTH	:	
		03	A7	9E	0006C	ADDL2	R2, LENGTH	:	
7E			04	C7	00070	MOVAB	3(R7), R0	:	2490
	00000000G		01	FB	00074	DIVL3	#4, R0, -(SP)	:	
			50	D0	0007B	CALLS	#1, DBG\$GET_TEMPMEM	:	
68			57	28	0007E	MOVL	R0, NEW_STRING	:	
			01	CE	00082	MOVC3	LENGTH, -(STRING), (NEW_STRING)	:	2491
			12	11	00085	MNEGL	#1, I	:	2492
	61	8F	6048	91	00087	BRB	9\$	:	
			0B	1F	0008C	CMPB	(I)[NEW_STRING], #97	:	2493
	7A	8F	6048	91	0008E	BLSSU	9\$	:	
			04	1A	00093	CMPB	(I)[NEW_STRING], #122	:	
	6048		20	82	00095	BGTRU	9\$	:	
EA			57	F2	00099	SUBB2	#32, (I)[NEW_STRING]	:	2495
			57	B0	0009D	AOBLSS	LENGTH, I, 8\$	:	2493
	04	AE	58	D0	000A0	MOVW	LENGTH, STRING_DESC	:	2496
			0D	11	000A4	MOVL	NEW_STRING, STRING_DESC+4	:	2497
	04	AE	51	D0	000A6	BRB	11\$	:	2481
						MOVL	R1, STRING_DESC+4	:	2504



51	04	A0	51	C3	000AA	SUBL3	R1, 4(R0), R1	2506
6E		51	60	A1	000AF	ADDW3	(R0), R1, STRING_DESC	
			02	DD	000B3	PUSHL	#2	2513
		00000000'	EF	9F	000B5	PUSHAB	P.AAQ	
		08	AE	9F	000BB	PUSHAB	STRING_DESC	
00000000G	00		03	FB	000BE	CALLS	#3, DBG\$NMATCH	
	01		50	D1	000C5	CMPL	R0, #1	
			06	12	000C8	BNEQ	12\$	
	0C	BC	02	D0	000CA	MOVL	#2, @TOKEN	2514
			3E	11	000CE	BRB	15\$	
		00000000'	02	DD	000D0	PUSHL	#2	2516
		08	EF	9F	000D2	PUSHAB	P.AAR	
00000000G	00		AE	9F	000D8	PUSHAB	STRING_DESC	
	01		03	FB	000DB	CALLS	#3, DBG\$NMATCH	
			50	D1	000E2	CMPL	R0, #1	
	0C	BC	06	12	000E5	BNEQ	13\$	
			03	D0	000E7	MOVL	#3, @TOKEN	2517
		00000000'	21	11	000EB	BRB	15\$	
		08	01	DD	000ED	PUSHL	#1	2519
			EF	9F	000EF	PUSHAB	P.AAS	
00000000G	00		AE	9F	000F5	PUSHAB	STRING_DESC	
	01		03	FB	000F8	CALLS	#3, DBG\$NMATCH	
			50	D1	000FF	CMPL	R0, #1	
	0C	BC	06	12	00102	BNEQ	14\$	
			09	D0	00104	MOVL	#9, @TOKEN	2520
	0C	BC	04	11	00108	BRB	15\$	
		00000000G	01	D0	0010A	MOVL	#1, @TOKEN	2523
	07		00	91	0010E	CMPL	DBG\$GB_LANGUAGE, #7	2527
			0B	12	00115	BNEQ	16\$	
50	04	AE	58	C3	00117	SUBL3	NEW STRING, STRING_DESC+4, R0	2530
5A		50	56	C1	0011C	ADDL3	POINTER, R0, TOKEN_END	
			3B	11	00120	BRB	22\$	2529
	5A	04	AE	D0	00122	MOVL	STRING_DESC+4, TOKEN_END	2533
			35	11	00126	BRB	22\$	2453
			51	D4	00128	CLRL	R1	2536
	30		59	91	0012A	CMPL	CHAR, #48	
			02	1F	0012D	BLSSU	18\$	
			51	D6	0012F	INCL	R1	
			50	D4	00131	CLRL	R0	
	39		59	91	00133	CMPL	CHAR, #57	
			02	1A	00136	BGTRU	19\$	
			50	D6	00138	INCL	R0	
	52		51	D2	0013A	MCOML	R1, R2	
	50		52	CA	0013D	BICL2	R2, R0	
	01		50	D1	00140	CMPL	R0, #1	
			1A	12	00143	BNEQ	23\$	
	30		59	91	00145	CMPL	CHAR, #48	2538
			0C	1F	00148	BLSSU	21\$	
	39		59	91	0014A	CMPL	CHAR, #57	
			07	1A	0014D	BGTRU	21\$	
			56	D6	0014F	INCL	POINTER	
	59		66	90	00151	MOVB	(POINTER), CHAR	
			EF	11	00154	BRB	20\$	
	5A		56	D0	00156	MOVL	POINTER, TOKEN_END	2541
			06	D0	00159	MOVL	#6, @TOKEN	2542
	0C	BC	65	11	0015D	BRB	31\$	2453
			50	D4	0015F	CLRL	R0	2545

41	8F	59	91	00161	CMPB	CHAR, #65	:
		02	1F	00165	BLSSU	24\$	:
		50	D6	00167	INCL	R0	:
		52	D4	00169	CLRL	R2	:
5A	8F	59	91	0016B	CMPB	CHAR, #90	:
		02	1A	0016F	BGTRU	25\$	:
		52	D6	00171	INCL	R2	:
	51	50	D2	00173	MCOML	R0, R1	:
	52	51	CA	00176	BICL2	R1, R2	:
		51	D4	00179	CLRL	R1	2546
61	8F	59	91	0017B	CMPB	CHAR, #97	:
		02	1F	0017F	BLSSU	26\$	:
		51	D6	00181	INCL	R1	:
		50	D4	00183	CLRL	R0	:
7A	8F	59	91	00185	CMPB	CHAR, #122	:
		02	1A	00189	BGTRU	27\$	:
		50	D6	0018B	INCL	R0	:
	53	51	D2	0018D	MCOML	R1, R3	:
	50	53	CA	00190	BICL2	R3, R0	:
	50	52	C8	00193	BISL2	R2, R0	:
	01	50	D1	00196	CMPL	R0, #1	2545
		25	12	00199	BNEQ	30\$	:
	2C	59	91	0019B	CMPB	CHAR, #44	2548
		17	13	0019E	BEQL	29\$	:
5C	8F	59	91	001A0	CMPB	CHAR, #92	2550
		11	13	001A4	BEQL	29\$	:
	20	59	91	001A6	CMPB	CHAR, #32	2552
		0C	13	001A9	BEQL	29\$	:
	0D	59	91	001AB	CMPB	CHAR, #13	2554
		07	13	001AE	BEQL	29\$	:
		56	D6	001B0	INCL	POINTER	2557
	59	66	90	001B2	MOVB	(POINTER), CHAR	2558
		E4	11	001B5	BRB	28\$	2548
	5A	56	D0	001B7	MOVL	POINTER, TOKEN_END	2561
0C	BC	05	D0	001BA	MOVL	#5, @TOKEN	2562
		04	11	001BE	BRB	31\$	2453
0C	BC	01	D0	001C0	MOVL	#1, @TOKEN	2567
	EF	5B	D0	001C4	MOVL	TOKEN_START, LEX_STRING_DESC+4	2575
00000000' EF	00000000' SA	5B	A3	001CB	SUBW3	TOKEN_START, TOKEN_END, LEX_STRING_DESC	2576
		04	001D3	RET			2580

; Routine Size: 468 bytes, Routine Base: DBG\$CODE + 1317

; 2471 2581 1  
; 2472 2582 1



```
2474 2583 1 GLOBAL ROUTINE DBG$NPARSE_SCOPE_LIST (INPUT_DESC, SCOPE_LIST, MESSAGE_VECT) =
2475 2584 1
2476 2585 1 ++
2477 2586 1 FUNCTIONAL DESCRIPTION:
2478 2587 1
2479 2588 1 This routine parses the objects of a SET SCOPE command. The pathname
2480 2589 1 parser is called within a loop to parse each scope item. A longword vector
2481 2590 1 is constructed which contains the number of scope items in the first cell
2482 2591 1 with the addresses of pathname descriptors in the subsequent cells.
2483 2592 1
2484 2593 1 A limit of 50 scope items per SET SCOPE command is observed.
2485 2594 1
2486 2595 1 This routine supplies the address of SCOPE_SCANNER as the lexical
2487 2596 1 analyzer for the pathname parser.
2488 2597 1
2489 2598 1 FORMAL PARAMETERS:
2490 2599 1
2491 2600 1 INPUT_DESC - A longword containing the address of a standard
2492 2601 1 character string descriptor reflecting the input
2493 2602 1
2494 2603 1 SCOPE_LIST - The address of a longword to contain the address
2495 2604 1 of the pathname descriptor vector
2496 2605 1
2497 2606 1 MESSAGE_VECT - The address of a longword to contain the address
2498 2607 1 of a message argument vector on error
2499 2608 1
2500 2609 1 IMPLICIT INPUTS:
2501 2610 1
2502 2611 1 NONE
2503 2612 1
2504 2613 1 IMPLICIT OUTPUTS:
2505 2614 1
2506 2615 1 On success, the pathname descriptor vector is obtained.
2507 2616 1
2508 2617 1 On failure, a message argument vector is constructed and returned.
2509 2618 1
2510 2619 1 ROUTINE VALUE:
2511 2620 1
2512 2621 1 An unsigned integer longword completion code
2513 2622 1
2514 2623 1 COMPLETION CODES:
2515 2624 1
2516 2625 1 ST$K_SUCCESS (1) - Success. Pathname descriptor vector formed.
2517 2626 1
2518 2627 1 ST$K_SEVERE (4) - Failure. Error detected. Message argument vector
2519 2628 1 constructed.
2520 2629 1
2521 2630 1 SIDE EFFECTS:
2522 2631 1
2523 2632 1 If more than 50 scopes are collected, this routine will issue a string
2524 2633 1 truncation message.
2525 2634 1
2526 2635 1 --
2527 2636 2 BEGIN
2528 2637 2
2529 2638 2 MAP
2530 2639 2 INPUT_DESC : REF dbg$stg_desc;
```

```
2531 2640 2
2532 2641 3
2533 2642 3
2534 2643 3
2535 2644 3
2536 2645 3
2537 2646 3
2538 2647 3
2539 2648 3
2540 2649 3
2541 2650 3
2542 2651 3
2543 2652 3
2544 2653 3
2545 2654 3
2546 2655 3
2547 2656 3
2548 2657 3
2549 2658 3
2550 2659 3
2551 2660 3
2552 2661 3
2553 2662 3
2554 2663 3
2555 2664 3
2556 2665 3
2557 2666 3
2558 2667 3
2559 2668 3
2560 2669 3
2561 2670 3
2562 2671 3
2563 2672 4
2564 2673 4
2565 2674 4
2566 2675 4
2567 2676 4
2568 2677 4
2569 2678 4
2570 2679 4
2571 2680 4
2572 2681 4
2573 2682 4
2574 2683 4
2575 2684 4
2576 2685 4
2577 2686 5
2578 2687 4
2579 2688 4
2580 2689 4
2581 2690 4
2582 2691 4
2583 2692 4
2584 2693 4
2585 2694 4
2586 2695 4
2587 2696 4

LITERAL
SCOPE_VECT_SIZE      = 51,
MAX_NOM_SCOPES       = 50;

LOCAL
SCOPE_VECT            : REF VECTOR,    ! Pathname descriptor vector
INDEX                 ! Index into the vector
DUMMY1,               ! Dummy parameter
DUMMY2,
STATUS;               ! Return status from
                     ! the pathname parser.

! Allocate space for 50 pathname descriptor pointers, plus one for the count.
scope_vect = dbg$get_tempmem(scope_vect_size);

! Loop and collect the pathname descriptors
index = 1;
WHILE true
DO
BEGIN
! For language C, we do some fancy footwork to
! make sure we preserve the original casing of
! the identifiers (since casing is significant
! in C).
IF .dbg$gb_language EQL dbg$k_c
THEN
BEGIN
LOCAL
length,
new_pointer: REF VECTOR [,BYTE], ! Pointer to orig. command input
pointer,    ! Pointer into input string
stg_desc: dbg$stg_desc,          ! String descriptor
temp_ptr;

! Obtain a pointer into the current command buffer and check
! that it is still within the range of the start and end of
! the command buffer that we saved away in DBG$NGET_CMD.
pointer = .input_desc[dsc$a_pointer];
IF (.pointer LSS .dbg$gl_upcase_command_ptr[0]) OR
(.pointer GTR .dbg$gl_upcase_command_ptr[1])
THEN
$DBG_ERROR('DBGNPNP\DBG$NPARSE_SCOPE_LIST 10');

! Obtain a pointer into the original (not up-cased)
! command buffer (TEMP_PTR).
! Copy from this buffer into a new buffer pointed to
! by NEW_POINTER.
! We unfortunately have to allocate memory
! and copy strings in order to stuff a
! trailing carriage return at the end.
```



```
2588 2697 4      !
2589 2698 4      !length = .input_desc[dsc$w_length];
2590 2699 4      new_pointer = dbg$get_tempmem((.length+3)/4);
2591 2700 4      temp_ptr = (.pointer = .dbg$gl_upcase_command_ptr[0]) +
2592 2701 4      .dbg$gl_orig_command_ptr;
2593 2702 4      CH$MOVE (.length, .temp_ptr, .new_pointer);
2594 2703 4      new_pointer[.length-1] = dbg$k_car_return;
2595 2704 4
2596 2705 4      ! Fill in the string descriptor.
2597 2706 4      !
2598 2707 4      stg_desc[dsc$b_class] = dsc$k_class_s;
2599 2708 4      stg_desc[dsc$b_dtype] = dsc$k_dtype_t;
2600 2709 4      stg_desc[dsc$w_length] = .length;
2601 2710 4      stg_desc[dsc$a_pointer] = .new_pointer;
2602 2711 4      stg_desc[dsc$l_pos] = 0;
2603 2712 4
2604 2713 4      ! Pick up the pathname.
2605 2714 4      !
2606 2715 4      status = dbg$np pathname_parser ( stg_desc,
2607 2716 4      scope_scanner,
2608 2717 4      scope_vect [.index],
2609 2718 4      dummy1,
2610 2719 4      dummy2,
2611 2720 4      true);
2612 2721 4
2613 2722 4      ! Update the input descriptor.
2614 2723 4      !
2615 2724 4      input_desc[dsc$w_length] = .input_desc[dsc$w_length] -
2616 2725 4      (.length - .stg_desc[dsc$w_length]);
2617 2726 4      input_desc[dsc$a_pointer] = .input_desc[dsc$a_pointer] +
2618 2727 4      (.length - .stg_desc[dsc$w_length]);
2619 2728 4      END
2620 2729 4
2621 2730 4      ! All other languages besides C ...
2622 2731 4      !
2623 2732 3      ELSE
2624 2733 3
2625 2734 3      status = dbg$np pathname_parser (.input_desc,
2626 2735 3      scope_scanner,
2627 2736 3      scope_vect [.index],
2628 2737 3      dummy1,
2629 2738 3      dummy2,
2630 2739 3      true);
2631 2740 3
2632 2741 3      IF NOT .status
2633 2742 3      THEN
2634 2743 4      BEGIN
2635 2744 4      IF dbg$nmatch (.input_desc, UPLIT BYTE (1, dbg$k_car_return), 1)
2636 2745 4      THEN
2637 2746 5      BEGIN
2638 2747 5      .message_vect = dbg$nmake_arg_vect (dbg$_needmore);
2639 2748 5      RETURN sts$k_severe;
2640 2749 5      END
2641 2750 5
2642 2751 4      ELSE
2643 2752 5      BEGIN
2644 2753 5      .message_vect = dbg$nsyntax_error (dbg$next_word (.input_desc));
```

```
2645      2754      5      RETURN sts$k_severe;
2646      2755      4      END;
2647      2756      4
2648      2757      3      END;
2649      2758      3
2650      2759      3
2651      2760      3      ! Look for a comma that separates scopes
2652      2761      3      !
2653      2762      3      IF NOT dbg$nmatch (.input_desc, UPLIT BYTE (%ASCIC ','), 1)
2654      2763      3      THEN
2655      2764      3          EXITLOOP;
2656      2765      3
2657      2766      3      ! Check for end of line
2658      2767      3      !
2659      2768      3      IF dbg$nmatch (.input_desc, UPLIT BYTE (1, dbg$k_car_return), 1)
2660      2769      3      THEN
2661      2770      4          BEGIN
2662      2771      4              .message_vect = dbg$nmake_arg_vect (dbg$_needmore);
2663      2772      4              RETURN sts$k_severe;
2664      2773      3          END;
2665      2774      3
2666      2775      3      ! There is atleast one more scope. Check for exceeding the limit.
2667      2776      3      !
2668      2777      3      IF .index GEQ max_num_scopes
2669      2778      3      THEN
2670      2779      4          BEGIN
2671      2780      4              ! Issue a truncation message.
2672      2781      4              !
2673      2782      4              dbg$nout_info (dbg$_stgtrunc);
2674      2783      4
2675      2784      4
2676      2785      4              ! Set up a phony input descriptor and exit the loop
2677      2786      4              !
2678      2787      4              input_desc [dsc$a_pointer] = UPLIT BYTE (dbg$k_car_return);
2679      2788      4              input_desc [dsc$w_length] = 1;
2680      2789      4              EXITLOOP;
2681      2790      4          END;
2682      2791      3
2683      2792      3
2684      2793      3      ! Update the index
2685      2794      3      !
2686      2795      3      index = .index + 1;
2687      2796      3
2688      2797      3      END;          ! End of loop
2689      2798      3
2690      2799      3
2691      2800      3
2692      2801      3      ! The scopes have been collected. Set the count.
2693      2802      3      !
2694      2803      3      scope_vect [0] = .index;
2695      2804      3
2696      2805      3
2697      2806      3      ! Return the scope list and success
2698      2807      3      !
2699      2808      3      .scope_list = .scope_vect;
2700      2809      3
2701      2810      2      RETURN sts$k_success;
```



: 2702  
: 2703

2811 2  
2812 1

END;

```
.PSECT DBG$PLIT,NOWRT, SHR, PIC,0

50 4E 24 47 42 44 5C 50 4E 50 4E 47 42 44 20 0003E P.AAT: .ASCII \ DBGNPNP\<92>\DBG$NPARSE_SCOPE_LIST 10\
54 53 49 4C 5F 45 50 4F 43 53 5F 45 53 30 0004D
                                0005C
                                0D 01 0005F P.AAU: .BYTE 1, 13
                                2C 01 00061 P.AAV: .ASCII <1>\ \
                                0D 01 00063 P.AAW: .BYTE 1, 13
                                0D 00065 P.AAX: .BYTE 13

.PSECT DBG$CODE,NOWRT, SHR, PIC,0

                                OFFC 00000
                                5E 18 C2 00002
                                33 DD 00005
00000000G 00 01 FB 00007
                                5B 50 D0 0000E
                                59 01 D0 00011
                                57 04 AC D0 00014
                                5A 6B49 DE 00018 1$:
07 00000000G 00 91 0001C
                                03 13 00023
                                0092 31 00025
                                52 04 A7 D0 00028 2$:
00000000G 00 52 D1 0002C
                                09 19 00033
00000000G 00 52 D1 00035
                                15 15 0003C
                                00000000' EF 9F 0003E 3$:
                                01 DD 00044
                                00028362 8F DD 00046
00000000G 00 03 FB 0004C
                                56 67 3C 00053 4$:
50 50 03 A6 9E 00056
7E 50 04 C7 0005A
00000000G 00 01 FB 0005E
                                58 50 D0 00065
                                52 00000000G 00 C2 00068
                                52 00000000G 00 C1 0006F
                                60 56 28 00077
                                FF A648 0D 90 0007B
                                OE AE 010E 8F B0 00080
                                OC AE 56 B0 00086
                                10 AE 58 D0 0008A
                                14 AE D4 0008E
                                01 DD 00091
                                08 AE 9F 00093
                                10 AE 9F 00096
                                5A DD 00099

                                SUBL2
                                PUSHL #24, SP
                                CALLS #51, DBG$GET_TEMP_MEM
                                MOVL R0, SCOPE_VECT
                                MOVL #1, INDEX
                                MOVL INPUT_DESC, R7
                                MOVAL (SCOPE_VECT)[INDEX], R10
                                CMPB DBG$GB_LANGUAGE, #7
                                BEQL 2$
                                BRW 5$
                                MOVL 4(R7), POINTER
                                CMPL POINTER, DBG$GL_UPCASE_COMMAND_PTR
                                BLSS 3$
                                CMPL POINTER, DBG$GL_UPCASE_COMMAND_PTR+4
                                BLEQ 4$
                                PUSHAB P.AAT
                                PUSHL #1
                                PUSHL #164706
                                CALLS #3, LIB$SIGNAL
                                MOVZWL (R7), LENGTH
                                MOVAB 3(R6), R0
                                DIVL3 #4, R0, -(SP)
                                CALLS #1, DBG$GET_TEMP_MEM
                                MOVL R0, NEW_POINTER
                                SUBL2 DBG$GL_UPCASE_COMMAND_PTR, R2
                                ADDL3 DBG$GL_ORIG_COMMAND_PTR, R2, TEMP_PTR
                                MOVCL3 LENGTH-(TEMP_PTR)-(NEW_POINTER)-PTR
                                MOVAB #13, -1(LENGTH)[NEW_POINTER]
                                MOVW #270, STG_DESC+2
                                MOVW LENGTH, STG_DESC
                                MOVL NEW_POINTER, STG_DESC+4
                                CLRL STG_DESC+8
                                PUSHL #1
                                PUSHAB DUMMY2
                                PUSHAB DUMMY1
                                PUSHL R10
```



		FD8D	CF	9F	0009B	PUSHAB	SCOPE_SCANNER	2715
		20	AE	9F	0009F	PUSHAB	STG_DESC	
EA6E	CF		06	FB	000A2	CALLS	#6, DBG\$NPATHNAME_PARSER	2717
	6E		50	DO	000A7	MOVL	R0, STATUS	
	50	OC	AE	3C	000AA	MOVZWL	STG_DESC, R0	2725
	50		56	C2	000AE	SUBL2	LENGTH, R0	
	67		50	A0	000B1	ADDW2	R0, (R7)	
04	A7		50	C2	000B4	SUBL2	R0, 4(R7)	2727
			18	11	000B8	BRB	6\$	2670
			01	DD	000BA	PUSHL	#1	2736
		08	AE	9F	000BC	PUSHAB	DUMMY2	
		10	AE	9F	000BF	PUSHAB	DUMMY1	
			5A	DD	000C2	PUSHL	R10	
		FD64	CF	9F	000C4	PUSHAB	SCOPE_SCANNER	2734
			57	DD	000C8	PUSHL	R7	2736
EA46	CF		06	FB	000CA	CALLS	#6, DBG\$NPATHNAME_PARSER	
	6E		50	DO	000CF	MOVL	R0, STATUS	
	28		6E	E8	000D2	BLBS	STATUS, 7\$	2741
			01	DD	000D5	PUSHL	#1	2744
		00000000'	EF	9F	000D7	PUSHAB	P.AAU	
			57	DD	000DD	PUSHL	R7	
00000000G	00		03	FB	000DF	CALLS	#3, DBG\$NMATCH	
	3C		50	E8	000E6	BLBS	R0, 8\$	
			57	DD	000E9	PUSHL	R7	2753
00000000G	00		01	FB	000EB	CALLS	#1, DBG\$NNEXT_WORD	
			50	DD	000F2	PUSHL	R0	
00000000G	00		01	FB	000F4	CALLS	#1, DBG\$NSYNTAX_ERROR	
			35	11	000FB	BRB	9\$	
			01	DD	000FD	PUSHL	#1	2762
		00000000'	EF	9F	000FF	PUSHAB	P.AAV	
			57	DD	00105	PUSHL	R7	
00000000G	00		03	FB	00107	CALLS	#3, DBG\$NMATCH	
	4D		50	E9	0010E	BLBC	R0, 12\$	
			01	DD	00111	PUSHL	#1	2768
		00000000'	EF	9F	00113	PUSHAB	P.AAW	
			57	DD	00119	PUSHL	R7	
00000000G	00		03	FB	0011B	CALLS	#3, DBG\$NMATCH	
	15		50	E9	00122	BLBC	R0, 10\$	
		000280D0	8F	DD	00125	PUSHL	#164048	2771
00000000G	00		01	FB	0012B	CALLS	#1, DBG\$NMAKE_ARG_VECT	
	OC		50	DO	00132	MOVL	R0, @MESSAGE_VECT	
	50		04	DO	00136	MOVL	#4, R0	2772
				04	00139	RET		
	32		59	D1	0013A	CMPL	INDEX, #50	2777
			1A	19	0013D	BLSS	11\$	
		0002804B	8F	DD	0013F	PUSHL	#163915	2783
00000000G	00		01	FB	00145	CALLS	#1, DBG\$NOUT_INFO	
	A7	00000000'	EF	9E	0014C	MOVAB	P.AAX, 4(R7)	2788
	67		01	B0	00154	MOVW	#1, (R7)	2789
			05	11	00157	BRB	12\$	2779
			59	D6	00159	INCL	INDEX	2796
			59	31	0015B	BRW	1\$	2661
			59	DO	0015E	MOVL	INDEX, (SCOPE_VECT)	2803
08	6B		5B	DO	00161	MOVL	SCOPE_VECT, @SCOPE_LIST	2808
	BC		01	DO	00165	MOVL	#1, R0	2810
	50			04	00168	RET		2812



DBGPNP  
V04-000

D 8  
16-Sep-1984 01:50:44  
14-Sep-1984 12:17:18

VAX-11 Bliss-32 V4.0-742  
[DEBUG.SRC]DBGPNP.B32;1

Page 94  
(25)

; Routine Size: 361 bytes, Routine Base: DBG\$CODE + 14EB

: 2705 2813 1 END  
: 2706 2814 0 ELUDOM

!End of module

.EXTRN LIB\$SIGNAL

PSECT SUMMARY

Name	Bytes	Attributes
DBG\$OWN	68	NOVEC, WRT, RD, NOEXE, NOSHR, LCL, REL, CON, PIC, ALIGN(2)
DBG\$PLIT	102	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)
DBG\$CODE	5716	NOVEC, NOWRT, RD, EXE, SHR, LCL, REL, CON, PIC, ALIGN(0)

Library Statistics

File	Total	Symbols Loaded	Percent	Pages Mapped	Processing Time
-\$255\$DUA28:[SYSLIB]LIB.L32;1	18619	9	0	1000	00:01.9
-\$255\$DUA28:[DEBUG.OBJ]STRUCDEF.L32;1	32	0	0	7	00:00.1
-\$255\$DUA28:[DEBUG.OBJ]DBGLIB.L32;1	1545	32	2	97	00:02.1
-\$255\$DUA28:[DEBUG.OBJ]DSTRECRDS.L32;1	418	1	0	31	00:00.3
-\$255\$DUA28:[DEBUG.OBJ]DBGMSG.L32;1	386	4	1	22	00:00.3

COMMAND QUALIFIERS

: BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:DBGNPNP/OBJ=OBJ\$:DBGNPNP MSRC\$:DBGNPNP/UPDATE=(ENH\$:DBGNPNP)

: Size: 5716 code + 170 data bytes  
: Run Time: 01:38.3  
: Elapsed Time: 04:09.9  
: Lines/CPU Min: 1718  
: Lexemes/CPU-Min: 21243  
: Memory Used: 380 pages  
: Compilation Complete



0087 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

DBGNMSG  
LIS

DBGNHELP  
LIS

DBGNPARSE  
LIS

DBGNEXCTE  
LIS

DBGNPNP  
LIS



0088 AH-BT13A-SE  
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION  
CONFIDENTIAL AND PROPRIETARY

